TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUUSO PISKONEN

DATAMIGRAATIO TYÖTIETOKANNAN AVULLA

Diplomityö

Tarkastaja:
Yliopistonlehtori Timo Aaltonen
Tarkastaja ja aihe hyväksytty
31. maaliskuuta 2017

TIIVISTELMÄ

JUUSO PISKONEN: Datamigraatio työtietokannan avulla
Tampereen teknillinen yliopisto
Diplomityö, 42 sivua
Toukokuu 2017
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Timo Aaltonen
Avainsanat: datamigraatio, prosessiuudistus, työtietokanta

Datamigraatiossa dataa siirretään järjestelmien välillä muokaten sitä kohdejärjestelmään sopivaksi. Prosessi voidaan jakaa kolmeen osaan: Datan hakeminen lähdejärjestelmästä, datan muokkaus ja vienti kohdejärjestelmään. Lähdedata luetaan käsittelyä varten, muokataan prosessointivaiheessa kohdejärjestelmän vaatimaan muotoon ja kirjoitetaan kohdejärjestelmään.

Työssä uudistettiin toistuva datamigraatioprosessi, joka oli toteutettu aiemmin Java-ohjelmointikielellä käyttäen Spring Batch -ohjelmistokehystä. Prosessi koostui useista sovelluksella toteutetuista eräajoista, jotka lukivat datan kahdesta eri tietolähteestä, käsittelivät sen ja lopuksi kirjoittivat sovelluksen käyttämään tietokantaan. Prosessissa oli ongelmana pitkä suoritusaika, korkea resurssien tarve ja haastava toistettavuus.

Datamigraatioprosessi uudistettiin käyttämällä datan prosessointiin ja muokkaukseen työtietokantaa. Työtietokantana toimi PostgreSQL -tietokanta, jonka ominaisuuksista hyödynnettiin varsinkin näkymiä ja viitetauluja, joista viitetauluja käytettiin datan siirrossa sovelluksen tietokantaan.

Työtietokannan käyttöönotto datan prosessoinnissa vei suurimman osan migraatioprosessin vaatimista resursseista sovelluksesta erilliseen tietokantaan. Prosessin kokonaiskesto laski huomattavasti ja työtietokantaan tehdyt näkymät mahdollistivat datan katselmoinnin jo kantatasolla.

ABSTRACT

JUUSO PISKONEN: Data Migration with a Staging Database
Tampere University of Technology
Master of Science Thesis, 42 pages
May 2017
Masters Degree Programme in Information Technology
Major: Software Engineering
Examiner: Timo Aaltonen
Keywords: data migration, staging database

Data migration is a process that moves data from one system to another while processing it to be suitable for the destination system. The process can be divided to three phases: reading data from the source system, processing the data and writing the data to the destination system. In the process, data is read from the source system to be processed to be suitable for the destination system and then wrote to the system.

In this thesis, the data migration process was renewed by replacing the old migration process, written in Java, with usage of a staging database. The process was previously done with an application consisting multiple batch jobs made with Spring Batch framework, which read the data from two different data sources, processed the data in the application and wrote the processed data to the applications database. The process had problems with long execution time, high need of computation capacity and challenges with update process.

The renewed data migration process includes a staging database which was used in the data processing phase. The staging database used PostgreSQL's views and string functions to process the data to be used as objects by the application. After the processing, the data was copied to the applications database by using the PostgreSQL's data wrapper and foreign tables which made possible to copy the data in the database layer.

The staging database handled well all the data processing, which previously took most of the time and computation capacity. The total duration of the migration process fell significantly and the views used to process the data made possible to review the data more efficiently.

ALKUSANAT

Diplomityö pohjautuu ADA Drive Oy:n sisäisen datamigraatioprosessin uudistukseen. Prosessiuudistus kehitettiin ja tehtiin tarpeeseen, ja vasta sen jälkeen siinä havaittiin olevan potentiaalia myös diplomityöhön. Mukana prosessia uudistamassa oli ADA:lta Jussi Marttila ja Jani Santanen, jotka vaikuttivat alusta asti uudistuksen etenemiseen ja lopputulokseen.

Tampereen teknilliseltä yliopistolta diplomityötä ohjasi yliopistonlehtori Timo Aaltonen, jota haluan kiittää rakentavasta palautteesta ja pitkästä pinnasta työn valmistumisen pitkittyessä. Aina ei vain aika meinannut riittää.

Lopuksi haluan kiittää minua työn aikana tukeneita: Perheelleni kiitos, että jaksoitte kysellen muistuttaa minua valmistumisesta ja tutkinnon tärkeydestä. Urrrheilujoukkue NMKSV:lle kiitos vertaistuesta, kannustuksesta ja lohdutuksesta tuskaisina hetkinä. Kiitos Tampereen TietoTeekkarikilta ilmaisesta kahvista ja kiltahuoneesta, joka toimi toisena kotina vieraillessani Tampereella. Kiitos kaikille ystäville ja työkavereille, jotka muistuttivat välillä rauhoittua ja nauttia elämästä, myös tämän työn aikana. Erityiskiitokset avovaimolleni Millalle, joka myös toimi tarvittaessa työni oikolukijana.

Helsingissä, 18.4.2017

Juuso Piskonen

SISÄLLYS

KUVALUETTELO

TAULUKKOLUETTELO

OHJELMALUETTELO

TERMIT JA NIIDEN MÄÄRITELMÄT

BASE	ADA Drive Oy:n tuote, joka sisältää moniosaisen tuotepaketin ajoneuvohuoltamoille suunnattuja tuotteita, kuten kuluttajapalvelun, mobiilipalvelun ja huoltamon työnjohto-ohjelmiston.
Datamigraatioprosessi	Datan siirto- ja muokkausprosessi, jossa data siirretään ja muokataan uuteen järjestelmään sopivaksi.
Datan rikastaminen	Tiedon yhdistäminen ja muokkaaminen, jotta sen tietoarvo tai oikeellisuus paranee.
Datan suodattaminen	Datasta suodatetaan eli tiputetaan pois osia ennalta päätettyjen kriteereiden mukaisesti.
JSON	JavaScript Object Notation on avoimen standardin tiedostomuoto tiedonvälitykseen.
Java Spring	Ohjelmistokehys, joka sisältää laajan valikoiman apuluokkia ja kirjastoja.
JavaBean	Uudelleen käytettävä ohjelmakomponentti, joita käytetään tiedon käsittelyyn.
Java	Ohjelmointikieli, jota ei käännetä konekieleksi, vaan jota ajetaan virtuaalikoneessa.
Kantaluokka	Luokka, jonka toinen luokka perii muuttaen tai lisäten sen alkuperäistä toiminnallisuutta.
Lueteltu tyyppi	Muuttuja, jonka kaikki mahdolliset arvot ovat lueteltu muuttujan rakenteessa.
Oliopohjainen sovellus	Ohjelma, jonka toteutuksessa on käytetty lähestymistapana olio-ohjelmointia.
Pilvipalvelu	Hajautettu internetissä tarjottava palvelu.
PostgreSQL	Suosittu avoimen lähdekoodin relaatiotietokanta.
REST	Representational State Transfer on varsinkin verkkosovelluksissa käytetty arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.

Rajapinta	Ohjelmointirajapinta on määritelmä, jonka avulla eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään.
Relaatiotietokanta	Tietokanta, joka perustuu predikaattilogiikkaan pohjautuvaan relaatiomalliin.
Skeema	Tietokannassa määrittelykokonaisuus, joka voi sisältää tauluja tai näkymiä.
Spring Batch	Ohjelmistokehys eräajoprosesseihin.
TecDoc	Ajoneuvojen varaosiin ja niiden yhteensopivuuksiin keskittynyt tietopalvelu.
Tietokanta	Tietokanta on tietokannan hallintajärjestelmällä hallittava joukko loogisesti toisiinsa liittyvää tietoa.ğ
Totuusarvo	Muuttuja, joka voi saada arvokseen tosi tai epätosi.
Verkkopalvelu	Internetissä tarjottava palvelu.
Vierasavain	Attribuutti tai attribuuttiyhdistelmä, joka osoittaa relaatiotietokannassa johonkin toiseen tietokantatauluun ja yksilöi sieltä rivin.

1. JOHDANTO

Pilvipohjaiset verkkopalvelut ovat yleistyneet viime vuosina nopeiden internet yhteyksien ja pilvipohjaisen laskennan ansiosta. Tämä takaa palveluille mahdollisuuden suurempiin tietomääriin ja datan käyttämiseen laskennan tukena. Pilvipohjaiset verkkopalvelut kilpailevat perinteisten työpöytäohjelmistojen kanssa ylläpidettävyyden ja hinnan avulla [1, 1-3].

ADA Drive Oy tarjoaa kilpailijan perinteisille autohuoltamo-ohjelmistoille pilvipohjaisella *BASE-ratkaisullaan* [2]. Ohjelmistossa yhdistyy autohuoltamon käyttämä työnjohtopuoli, kuluttajille tarjottava /textithuoltolaskuri ajanvarauksella sekä kuluttajille suunnattu *mobiilisovellus*, josta kuluttaja pystyy seuraamaan ajoneuvonsa määräaikaishuoltoja ja varaamaan ajan seuraavaan määräaikaishuoltoon. BASE-ratkaisu pohjautuu taustalla toimivaan tietopalveluun, joka yhdistää kolmannen osapuolen ajoneuvohaun (Suomessa TraFi, Ruotsissa Infobil, Norjassa Vegvesen), ajoneuvon mallinnuksen, ajoneuvojen huoltotiedon ja TecDoc varaosatiedon tarjotakseen ajoneuvoille yhteensopivat määräaikaishuollot, varaosat, renkaat ja huoltohistorian [3] [4] [5] [6]. Palvelun toteutuksessa on käytetty laajaa autoalan osaamista yhdessä modernien teknologioiden ja ohjelmistoalan osaamisen kanssa. Taustalla toimivan tietopalvelun lähtökohtana on eri tietolähteiden datan yhdistäminen ja käyttö rinnakkain, jotta BASE-ratkaisulle voidaan tarjota mahdollisimman kattava ja oikeellinen tietosisältö ajoneuvon, huoltojen ja varaosien osalta.

Tietolähteiden tarjoamien datamassojen yhdistäminen järjestelmän käyttöön vaatii useita *datamigraatioprosesseja*, joissa data prosessoidaan järjestelmän vaatimaan muotoon ja yhdistetään mahdollisuuksien mukaan automaattisesti muista tietolähteistä saatuihin datoihin. Osan datamigraatioprosesseista voidaan ajatella olevan jatkuvia, sillä järjestelmän dataa korjataan ja parannetaan autoalan asiantuntijoiden avulla päivittäin tähän kehitetyillä, sisäiseen käyttöön tarkoitetuilla, työkaluilla. Tässä työssä keskitytään kahden tietolähteen toistuvaan datamigraatioprosessiin ja sen parantamiseen.

Datamigraatioprosessi on aiemmin suoritettu sovelluksen eräajona, jossa tieto luetaan lähdetietokannoista, käsitellään sovelluksessa ja tallennetaan sovellustietokan-

taan. Tämä prosessi on kestoltaan lähes vuorokauden mittainen, jonka aikana sovelluksen resurssit ovat prosessin käytössä, jolloin sovellus ei pysty vastaamaan normaalikäyttöön liittyviin rajapintakyselyihin tarpeeksi tehokkaasti. Lisäksi pitkä kesto altistaa helpommin prosessinaikaisille virheille. Prosessiuudistuksen tavoitteena on lyhentää datamigraatioprosessin kestoa, siirtää resurssien kulutus pois sovelluspalvelimelta sekä mahdollistaa käsitellyn datan jäljitettävyys tietolähteiden *raakadatoihin*.

Uutena yrityksen sisäisenä innovaationa datamigraatioprosessissa hyödynnetään *työtietokantaa*, jonka avulla datan prosessointi voidaan hoitaa ilman käytössä olevan järjestelmän rasittamista. Uudessa prosessimallissa raakadata luetaan työtietokantaan, jossa se prosessoidaan valmiiksi siirrettäväksi sovellustietokantaan. Työtietokannan käytön lisäksi hyödynnetään PostgreSQL -tietokannan ominaisuuksia, kuten *viitetauluja*, prosessoidun datan siirtämiseen varsinaiseen järjestelmään suoraan tietokantatasolla. Tämän tulisi taata nopeat siirtoajat, entistä lyhyemmät käyttökatkot ja järjestelmän minimaalisen kuormituksen prosessin aikana.

Tässä työssä esitellään datamigraatioprosessin uudistaminen, jossa aiemmin sovelluksessa suoritettu prosessointi uudistetaan käyttämään *migraatioalustana* tietokantaa. Uudistuksen jälkeen datamigraatioprosessia verrataan aiempaan ja uudistuksen hyötyjä analysoidaan. Työn rakenne esittelee aluksi datamigraatioprosessin ja työssä käytetyt teknologiat, jonka jälkeen pureudutaan uudistettavaan datamigraatioprosessiin ja sen heikkouksiin. Uusi datamigraatioprosessi esitellään 4. luvussa, jota seuraa prosessien arviointi ja vertailu. Lopuksi työn anti kootaan yhteen yhteenvedossa.

2. DATAMIGRAATIO JA TIETOKANNAN ERIKOISPIIRTEET

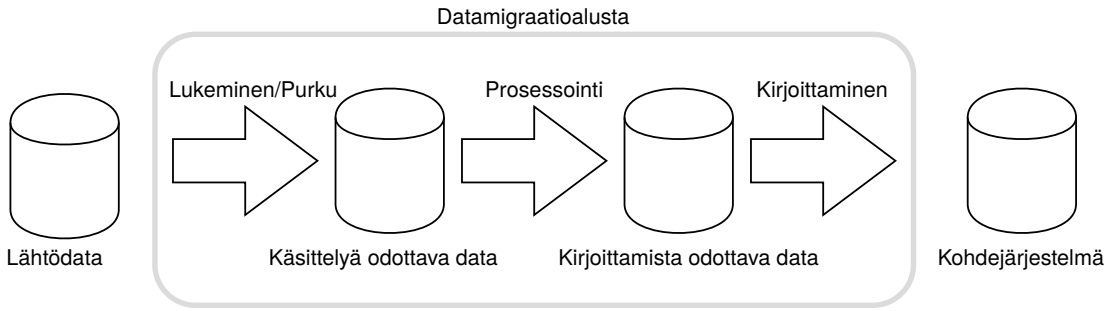
Datamigraatio on prosessi, jossa data otetaan järjestelmästä, muokataan ja viedään toiseen järjestelmään. Tässä kappaleessa esitellään prosessi, siihen liittyvät vaiheet ja migraatioon tässä projektissa käytettyjä työkaluja, kuten käytetyn tietokannan erityispiirteitä.

Datamigraatio voi tarkoittaa joko *datavaraston migraatiota* tai *sovellusdatan migraatiota*. Datavaraston migraatiossa on kyse tietokannan siirtämisestä tai päivittämisestä, joka voi koskea koko tietokantapalvelinta tai pelkkää tietokantaohjelmistoa. Sovellusdatan migraatiossa datan siirron lisäksi datan rakennetta muokataan prosessin aikana kohdejärjestelmään soveltuvaksi [7, 1-2]. Yleisesti sovellusdatan migraatioprosessi suoritetaan kerran järjestelmää päivitettäessä tai uuteen järjestelmään siirryttäessä. Tämä diplomityö keskittyy sovellusdatan migraatioon, mutta perinteisestä sovellusdatan migraatiosta poiketen, otetaan huomioon myös migraatioprosessin toistettavuus ja suorituskyky.

2.1 Datamigraatioprosessi

Datamigraatioprosessi voidaan jakaa kolmeen osaan. Datan hakeminen lähdejärjestelmästä, datan muokkaus ja vienti kohdejärjestelmään. Jokainen prosessin osa on projektiriippuvainen, sillä lähde- ja kohdejärjestelmiä on useita erilaisia. Järjestelmien lisäksi datamigraatioprosessin toteutukseen vaikuttavat käytetyt teknologiat ja datan määrä.

Datamigraation prosessimalli jakautuu kuvan 2.1 mukaisesti kolmeen osaan: Datan lukeminen, prosessointi ja kirjoittaminen kohdejärjestelmään. Tämä toteuttaa ETL-mallin (extract, transform, load) prosessin, jota käytetään monimuotoisen lähdedatan saattamiseen halutussa muodossa kohdejärjestelmään [8]. Lähdedata luetaan käsittelyä varten halutusta lähteestä. Lähteitä voivat olla rakenteiset dokumentit (XML-tiedostot), tekstitiedostot (CSV-tiedosto) tai tietokannat. Lukeminen voi tapahtua koko datamäärä kerrallaan tai osissa, jos kyseessä on useampaan tiedostoon



Kuva 2.1 *Datamigraation prosessimalli.*

jaettu lähdedata tai tietokanta. Lukemiseen voidaan käyttää valmiita komponentteja.

Prosessointivaiheessa lähdedatan rakennetta muokataan sovellukseen sopivaksi. Prosessointi suoritetaan ohjelmallisesti tavoitteena muokata lähtödatan rakenne sellaiseksi, että kohdesovellus pystyy käyttämään sitä. Esimerkiksi oliopohjaisessa sovelluksessa lähdedatasta muodostetaan kohdesovelluksen käyttämiä olioita. Prosessointivaihe voi sisältää *datan rikastamista*, jonka aikana lähdedataan yhdistetään itse tuotettua tai muualta johdettua tietoa [9, 3-5]. Kirjoitusvaiheessa prosessoitu data kirjoitetaan sovelluksen käyttämään tietokantaan. Kirjoitusprosessi voi sisältää *datan suodattamista*, jossa voidaan jättää vääräksi tai epäluotettavaksi havaittua tietoa kirjoittamatta kohdejärjestelmään. Tämä voi tarkoittaa esimerkiksi prosessoituja tietoja, jotka eivät ole korjaantuneet rikastamisprosessissa, joten suodattaminen on voidaan suorittaa kirjoitusprosessissa.

2.2 PostgreSQL

PostgreSQL on avoimella lähdekoodilla toteutettu *relaatiotietokanta* [10]. Relaatiotietokanta on tietovarasto, joka perustuu ensimmäisen kertaluvun predikaattilogiikkaan. Mallissa tietokantaan tallennettava data esitetään joukkona äärellisiä monikoita, jotka ovat ryhmitelty relaatioiksi tietokantatauluihin.

Relaatiotietokantoja on saatavilla useita, joiden joukosta löytyy sekä avoimen, että suljetun lähdekoodin ohjelmistoja. PostgreSQL on avoimeen lähdekoodiin perustuva tietokanta, jota on kehitetty yli 15 vuotta. Se toimii useilla käyttöjärjestelmillä, kuten Linux, Windows ja useat Unix-pohjaiset käyttöjärjestelmät. PostgreSQL:ään on tehty ohjelmointirajapinnat kaikille yleisesti käytetyille ohjelmointikielille [10].

PostgreSQL:ää kehitetään yhteisövetoisesti. Siitä löytyy useita ominaisuuksia, jotka kilpailevista, maksullisista, tietokantaohjelmistoista puuttuvat, kuten PostgreSQL

versiossa 9.3 esitellyt viitetaulut [10] [11] [12]. Nämä ominaisuudet mahdollistavat tietokantaohjelmiston käytön uusiin käyttötarkoituksiin, kuten datan muokkaamiseen osana datamigraatioprosessia.

2.2.1 Näkymät ja materialisoidut näkymät

Näkymä on tietokantakyselyllä määritetty relaatio, jolle on annettu nimi. Kysely joudutaan tekemään joka kerta uudestaan, kun näkymän sisältöä tahdotaan tarkastella. Tämä tuottaa näkymään aina uusimman version koostettavasta lähdedatasta, mutta tekee siitä hitaan käyttää. Näkymä ei tarvitse muistia kuin itse tietokantakyselyyn ja nimeen, joten se on muistinkulutuksen kannalta materialisoitua näkymää parempi [10, c.5].

Materialisoidussa näkymässä tietokantakyselyllä määritetty joukko tallennetaan muistiin ja sille annetaan nimi. PostgreSQL -tietokannan materialisoidun näkymän sisältö ei päivity automaattisesti, vaan sen käyttöön on määritetty *REFRESH MATERIALIZED* VIEW -komento, jolla materialisoitu näkymä voidaan päivittää. Materialisoitu näkymä vie muistia, mutta on hakuoperaatioissa näkymää huomattavasti nopeampi [10, c.5].

Näkymän ja materialisoidun näkymän välisten erojen takia voidaan antaa käyttösuositus: Näkymä on hyvä, jos sisältöä on vähän tai jos se päivittyy useasti. Näkymällä on hitaampi hakuaika, joka täytyy ottaa huomioon, jos hakuja näkymään tulee useita. Materialisoitu näkymä on parempi suurempiin tietomääriin ja staattisempiin tietohakuihin.

2.2.2 Viitetaulut

Viitetaulut (Foreign table) ovat osa PostgreSQL:lle kehitettyä, ulkoisten tietojen sitomiseen (Foreign Data Wrappers) tarkoitettua ominaisuuspakettia, joka tarjoaa tuen liittää yksi PostgreSQL-tietokanta useisiin eri tietokantoihin tai järjestelmiin, kuten MySQL, SQLite ja Cassandra [10, F.33]. Tietojen sitomisella pystytään mallintamaan toisen tietokannan taulut viitetauluiksi omaan tietokantaan. Tällöin niissä toimivat lukuoperaatiot ja joissain tapauksissa jopa kirjoitusoperaatiot tietokantojen välillä.

Viitetaulun luomiseen tarvitaan yhteys viitetietokantaan (*CREATE SERVER* -komento), jonka tietojen sitomiseen määritetään tarvittava sitoja (*CREATE FOREIGN DATA WRAPPER* -komento). Yhteyden muodostamista varten tarvitaan käyttäjätunnuksen sitominen viitetietokannan käyttäjätunnukseen (*CREATE USER MAPPING*

-komento). Tämän jälkeen viitetaulu voidaan luoda *CREATE FOREIGN TABLE* -komennolla. PostgreSQL:n viitetaulu tukee luku- ja hakuoperaatioita, mutta yhdensuuntaisesta liikenteestä johtuen, se ei tue kirjoittamista. [10, c.54]

2.2.3 Merkkijonot

PostgreSQL:n merkkijonot sisältävät kolme tietotyyppiä, joihin voidaan varastoida merkkejä.

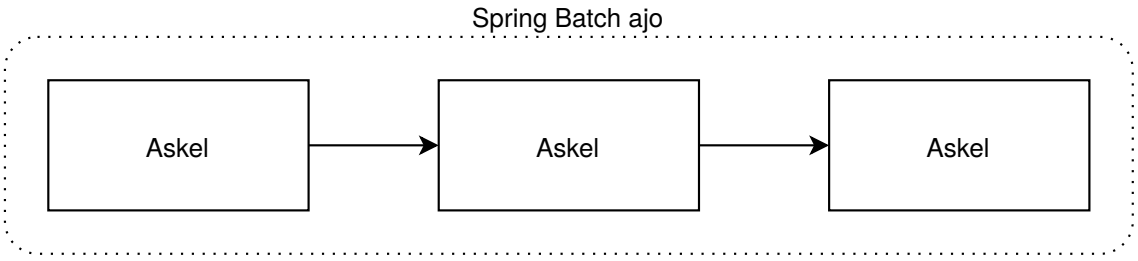
- Character varying (varchar)
- Character (char)
- Text

Character varying määrittelee merkkirajoituksen, joka rajoittaa merkkijonon pituuden yläpäästä, mutta ei aseta sille alarajaa. Character määrittelee täsmällisen merkkimäärän, johon merkkijono tarvittaessa täydennetään välilyönneillä automaattisesti. Käyttäjälle character-tyyppi kuitenkin näyttäytyy yhtenäisesti muiden merkkijonotyyppien kanssa, sillä se ei näytä täytevälejä käyttäjälle, eikä ota niitä huomioon tehtäessä vertailuoperaatioita kahden character-tyyppisen muuttujan välillä. Text -tyyppi ei ole SQL-standardin mukainen, sillä SQL-standardi ei sisällä vapaapituista merkkijonotyyppiä.[10, c.8.3]

Merkkijonofunktiot sisältävät kokoelman operaatioita ja funktioita merkkijonojen muokkaamiseen, merkkijonon tietojen tulostamiseen, kuten merkkijonon pituuden laskemiseen, sekä tyyppimuutoksiin. Merkkijonofunktioita voidaan käyttää hakuoperaatioiden yhteydessä, SELECT -lauseissa. Niillä voidaan muokata hakutuloksen merkkijonoja tai ottaa merkkijonojen muutoksia huomioon hakutermeissä.

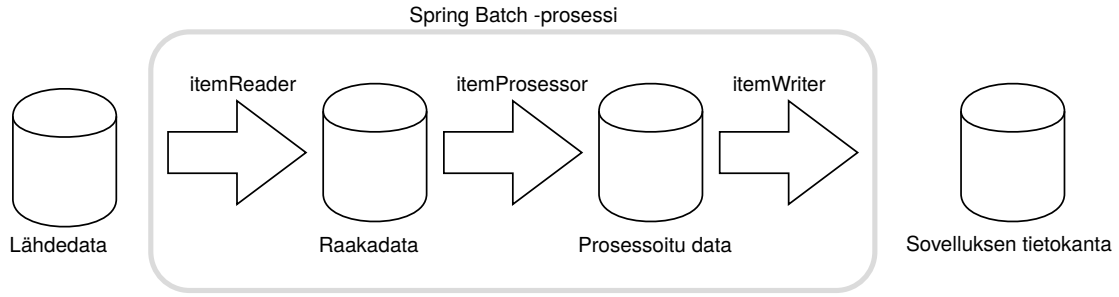
2.3 Spring Batch ja eräajo

Eräajo on prosessi, jossa syötettä luetaan, käsitellään ja tallennetaan osissa. Eräajo on usein käyttäjän vaikutuksesta riippumaton prosessi, joka käynnistymisen jälkeen suoriutuu valmiiksi itsenäisesti. Eräajoa käytetään yleisesti datamigraatioprosesseissa, sillä se kykenee käsittelemään suuria määriä dataa osissa. Sen itsenäinen ajotapa tarjoaa puitteet migraation ajamiseen ajastettuna silloin, kun laskuteho on halpaa, eikä se vie resursseja varsinaiselta käytöltä. [13]



Kuva 2.2 *Spring Batch ajo jakaantuu peräkkäisiin askeliin.*

Spring Batch on Accenturen ja SpringSourcen yhteistyössä tuottama kevyt ohjelmistokehys, jota ajetaan Java-virtuaalikoneessa (Java Virtual Machine, JVM). Spring Batch on kehitetty eräajon toteuttamiseen Spring -ohjelmistokehyksen rakenteen mukaisesti. Se on tarkoitettu yritystasoisten automaatioprosessien ajamiseen. Spring Batchin prosessirakenne koostuu askelista (step) ja niiden suorittamisessa halutussa järjestyksessä. Jokainen askel voi sisältää aliaskelia (slave step), jotka suoritetaan askeleen aikana. [14] Kuvan 2.2 mukaisesti astekeet ovat yhdistetty määrätyssä järjestyksessä toisiinsa, sillä kukin askel tietää mahdollisen seuraavan askeleen, joka suoritetaan tehdyn askeleen jälkeen. Kukin askel voi olla dataa lukeva, suodattava, prosessoiva tai kirjoittava.



Kuva 2.3 *Spring Batch ajon prosessi koostuu kolmesta vaiheesta: Datan lukeminen, prosessointi ja kirjoittaminen.*

Spring Batch jakaantuu prosessikaaviossa (kuva 2.3) esitettyihin osiin. Aluksi lähdedata luetaan itemReader:n avulla sille annetusta lähteestä. Tämän jälkeen dataa prosessoidaan haluttuun muotoon itemProcessor:lla ja lopuksi se kirjoitetaan kohdetietokantaan itemWriter:llä. Osille voidaan antaa parametrina haluttu osakoko, joka käsitellään kerralla. Sopiva osakoko riippuu aineistosta ja käytössä olevista resursseista.

2.4 Amazon Web Services

Amazon Web Services (AWS) on Amazonin markkinanimi *pilvipohjaisille verkkopalveluille*. AWS sisältää laajan joukon etätietojenkäsittelyresurssien palveluita, jotka muodostavat Amazon.com :n tarjoaman *pilvipalvelualustan* [15]. AWS tarjoaa pilvipalveluidensa rinnalla ohjelmointirajapinnat instanssien pystyttämiseen, muokkaamiseen ja sulkemiseen. Pilvipalvelut pyörivät fyysisesti useissa eri palvelulokaatioissa (Availability Zones), joista valittavana on yhden tai usean lokaation palvelumuoto. Pilvipalveluille tunnusomaisesti AWS vähentää palvelun ylläpidollista kuormaa huolehtimalla verkko- ja palveluinfrastruktuurista palveluntarjoajana [16].

2.5 Ajoympäristö

Ajoympäristöllä tarkoitetaan AWS:ää ja sen sisältämiä palveluita, kuten Amazon Elastic Cloud Computing -instanssi (EC2) ja Amazon Relation Database Service-tietokanta (RDS). Projektin ajoympäristö jakaantuu kahteen osaan: Sovellusta ajetaan EC2-instanssissa Centos-käyttöjärjestelmässä pyörivässä Java Virtual Machine:ssä (JVM) ja tietokannat ovat RDS-instansseissa [17].

Taulukko 2.1 *Amazon Web Services resurssit*

Instanssi	Instanssin tyyppi	vCPU	ECU	GB	Levytila (GB)
Sovellus	r3.large	2	6.5	15	1x 32 SSD
Työtietokanta	db.m4.large	2	6.5	8	1000
Sovellustietokanta	db.t2.medium	2	2	4	200

Projektissa käytetyt AWS-resurssit ovat lueteltu taulukossa 2.1. Kuten näistä käy ilmi, työtietokannalle on varattu enemmän muistia (GB) ja laskentatehoa (virtuaaliprosessori vCPU ja EC2 Compute Unit ECU) kuin sovellustietokannalle. Työtietokanta tarvitsee muistia esimerkiksi näkymien vastausjoukkojen muodostamiseen. Sovellusinstanssi kuuluu muistipainotteiseen instanssityyppiin, sillä Spring Batch eräajot tallentavat käyttömuistiin käsiteltäviä arvoja nopeuttaakseen prosessia. Tämän lisäksi sovellus vaatii muistia ajoneuvojen mallintamisessa, jonka aikana muistiin voidaan ladata useita ajoneuvomalleja.

Tietokannan luku- ja kirjoitusnopeutta mitataan luku- ja kirjoituskertoina sekunnissa (IOPS). Mittayksikkönä käytetään kilotavua sekunnissa. Käytössä oleville SSD levyille yksi luku- tai kirjoitusoperaation koko on 256 kilotavua. Tätä pienemmät operaatiot yritetään yhdistää yhdeksi luku- tai kirjoitusoperaatioksi. [18, 696-697] Amazon tarjoaa RDS -tietokannoille jatkuvaksi luku- ja kirjoitusnopeudeksi 3 IOPS/1GB

ja hetkittäiseksi nopeudeksi 3000 IOPS. Koska jatkuva nopeus kasvaa tietokannan koon mukaisesti, työtietokannan tilaksi on valittu 1000GB, joka mahdollistaa suurimman sallitun jatkuvan luku- ja kirjoitusnopeuden.

3. NYKYINEN MIGRAATIOPROSESSI

Migraatioprosessin vanha toteutus on tehty Java-ohjelmointikielellä käyttäen Spring Batch -ohjelmistokehystä. Ohjelmistokehys tarjoaa tuen eräajolle, jolla käsitellään lukutietokannasta saatava aineisto ja tallennetaan se tuotantotietokantaan. Tässä luvussa esitellään prosessia ja sen osia tietolähteistä ohjelman toteutukseen. Aliluvussa Kesto ja toistettavuus arvioidaan ratkaisun toimivuutta ja kestävyyttä toistettavuuden kautta.

Tarve migraatiolle tulee aineistojen käyttötarkoituksesta, joka on tarjota ajoneuvon varaosatietoja yksinkertaistetussa muodossa REST-rajapinnan kautta. Alkuperäisessä rakenteessaan lähdetietokantojen toteutus ei tue tätä käyttötapaa, joten data muokataan muotoon, jossa sen käyttö ajoneuvon tietomallinnuksessa ja osien hakemisessa on tehokkaampaa. Lisäksi data-aineistoa yhdistetään molemmista lähteistä mahdollisimman laadukkaan ajoneuvodatan tuottamiseksi.

3.1 Tietolähteet

Tietolähteet eli lähdedata sisältää kaksi ajoneuvojen osatietoja kokoavaa tietokantaa. Molemmat tietolähteet ovat kaupallisia ja tarkasti lisenssoituja. Niiden data luovutetaan käyttöön eri muodoissa sopimuksen mukaan, joten tässä esitetty prosessi on yksittäinen esimerkki niiden käytöstä. Normaalitilanteissa tietolähteiden dataa voitaisiin käyttää alkuperäisessä muodossaan, mutta muun järjestelmän kompleksisuuden vuoksi tietolähteiden data muokataan paremmin järjestelmälle sopivaan muotoon.

3.1.1 TecDoc-varaosatietokanta

TecAlliancen tarjoama TecDoc varaosatietokanta on tarkoitettu moottoriksi varaosien etsimiseen tiettyyn ajoneuvoon [6]. Sen data muodostuu satojen eri toimittajien varaosatiedoista ja TecAlliancen muodostamista ajoneuvomalleista. Toimittajat

tarjoavat varaosakataloginsa yhteensopivuustietojen kera TecAlliancelle, joka koostaa niistä tietokannan, jonka kautta ajoneuvojen varaosia on mahdollista etsiä. Varaosatietokanta on saatavilla valmiina verkkokauppatoteutuksena, sisältäen valmiin käyttöliittymän tietojen selaamiseen, työpöytäohjelmistona ja tietokantavedoksena. Tietokanta sisältää monitasoisen mallin ajoneuvosta, jonka avulla voidaan manuaalisesti mallintaa haluttu ajoneuvo ja löytää sitä vastaava ajoneuvomalli. Ajoneuvomalli koostuu Taulukossa 3.1 esitetyistä malli- ja tyyppi-tason tiedoista, siten, että ajoneuvon malli sisältää useampia ajoneuvotyyppejä. Mallin ollessa esimerkiksi Opel Astra J 1.6, voi ajoneuvotyypit sisältää kyseisestä mallista sekä Turbo, että ecoFLEX -versiot. Ajoneuvomallin avulla voidaan etsiä siihen sopivia varaosia eri kategorioittain.

Tässä työssä tietolähteenä käytetään TecDoc-tietokannan tietokantavedosta, joka toimitetaan neljä kertaa vuodessa määrämittaisina tekstitiedostoina (Fixed-length file). Tiedostoja on useita kutakin tietokantataulua kohden, sillä taulujen rivimäärät ovat suurimmillaan 157 miljoonaa. Tiedostot luetaan aliluvussa 2.4 esiteltyyn työtietokantaan omiin tauluihinsa ja indeksoidaan lukunopeuden optioimiseksi Javalla tehdyllä ohjelmalla. Tietokannassa tauluja on yhteensä 105 kappaletta, joten lukuoperaatio kestää noin 2.5 tuntia.

Taulukko 3.1 *Työssä käytetyt TecDoc tietokantataulut.*

Taulu	Rivimäärä	Sisältö
035 Text Modules	181430	Käännöstekstit eri kielillä
050 Criteria Table	1861	Attribuutti alkioiden määritykset
051 Key Tables	298	Yhdistelmätaulu listausavaimista
052 Key Table Entries	7565	Listojen arvot ja vakiot
100 Manufacturer	2844	Ajoneuvojen ja niiden osien valmistajat
110 Vehicle Model Series	11149	Ajoneuvon malli-tason tiedot
120 Vehicle Types	31229	Ajoneuvotyypin tarkemmat tiedot
200 Article Table	1980457	Varaosien tiedot
400 Article Search Tree	157587788	Varaosien hakeminen

Kaikki tietolähteen tarjoamat tietokantataulut eivät ole tässä käyttötarkoituksessa kiinnostavia, vaan voimme keskittyä taulukossa 3.1 esitettyihin tauluihin. Koska varaosatietokanta on monikielinen, löytyvät käännöstekstit käännöstaulusta 035. Attribuutit ja niiden arvot ovat eritelty tauluihin 050, 051 ja 052, joita käytetään muista tauluista kaksiosaisella avaimella. Ajoneuvot voidaan mallintaa taulujen 100, 110 ja 120 avulla. Taulu 200 kokoaa tiedot kaikista järjestelmän varaosista, joita voidaan hakea taulun 400 hakupuulla. Taulu 400 on järjestelmän suurin ja se sisältää tarvittavat tiedot ajoneuvon yhdistämiseen siihen sopiviin varaosiin. Vierasavaimia ja tietokantarajoituksia ei työtietokantaan ajeta, sillä ne hidastavat ohjelmallista

tiedon hakua. Huomioitavat poikkeustilanteet, kuten puuttuvat rivit, otetaan huomioon eräajoprosessissa ohjelmallisesti. Tällaisia ovat esimerkiksi eri taulujen väliset puutteet, kuten TecDoc tietokantataulun 120 Vehicle Types ajoneuvon tietoja tarkentava rivi, jonka mallitason tiedot löytyvät kuitenkin taulusta 110 Vehicle Model Series.

3.1.2 DriveRight rengasdata

DriveRight on WheelWizards:n palvelu, joka tarjoaa ajoneuvojen renkaisiin ja vanteisiin liittyviä tuotetietoja [19] [20]. Datapalveluiden lisäksi DriveRight sisältää korimalli- sekä vannekuvia, joiden avulla asiakas voi tehdä oman vannevalitsimen [21]. Tietolähde tarjoaa dataa ajoneuvon korimalliin liittyen, sisältäen renkaiden ja vanteiden valitsemiseen tarvittavat tiedot, kuten painon, maksiminopeuden, pulttijaon ja vannekiinnityksen leveyden. Ajoneuvotietojen lisäksi dataa löytyy vannemalleista, niiden koosta ja vanteisiin sopivista renkaista. DriveRight tarjoaa tiedon oman ajoneuvomallinsa yhdistämiseen TecDoc:n ajoneuvomalliin, joten molempien tietolähteiden data voidaan yhdistää käytettäväksi yhdessä järjestelmässä.

Tietolähteen tarjoamista datoista työssä keskitytään ajoneuvotietoihin ja niiden yhdistämiseen TecDoc:n tarjoamaan ajoneuvomalliin. Tähän soveltuvat taulut ovat esitelty taulukossa 3.2. Molemmista tietolähteistä löytyy yhtenäistä tietoa ajoneuvon korimallista, mutta DriveRight:n tarjoama renkaisiin liittyvä data on havaittu ADA Drive:n autoasiantuntijoiden analyysin perusteella paikkaansapitävämmäksi. Toisaalta TecDoc tarjoaa esimerkiksi korimallin tyypistä ja rakenteesta laadukkaampaa tietoa, joten molempien tietolähteiden prosessointi ja yhdistäminen tuottaa parhaan saatavilla olevan tietomallin ajoneuvon korimallista.

DriveRight tarjoaa sopimuksesta tietolähteen MS SQL Server -lukutietokantana [20]. Tietokanta sijaitsee WheelWizards:n palvelimella ja sinne tarjotaan käyttäjätunnus lukuoikeuksin sovittuihin tietokantatauluihin. Tietokantayhteys on auki sovituille IP-osoitteille, joten yhteys lukukantaan onnistuu vain sovelluspalvelimelta.

Taulukko 3.2 *Työssä käytetyt DriveRight tietokantataulut.*

Taulu	Rivimäärä	Sisältö
Tyrefit models	14441	Ajoneuvomallit
Tyrefit chassis	5754	Ajoneuvomallien korityypit
Ktype chassisId modelId	67525	TecDoc - DriveRight linkitys

Ajoneuvon mallinnukseen käytettävät tiedot löytyvät lukutietokannasta tauluista

Tyrefit models ja Tyrefit chassis. Taulut ovat yhdistettävissä toisiinsa suhteella monta yhteen, sillä useilla ajoneuvomalleilla on sama korimalli. DriveRight ajoneuvomalli pystytään yhdistämään TecDoc-tietokannan tarjoamaan ajoneuvomalliin Ktype chassisId modelId -taulun avulla, joka sisältää ktype:n eli TecDoc-ajoneuvomallin tunnisteen ja DriveRight:n käyttämät ajoneuvomallin ja korityypin tunnisteet.

3.2 Osa-ajo ja siirtoprosessi

Siirtoprosessi mukailee luvussa 2.3 esiteltyä Spring Batch ajon prosessimallia. Se on toteutettu Java Spring kehyksellä toteutetulla ohjelmistolla, joka sisältää oman data-moduulin, joka vastaa Spring Batch eräajoista. Poiketen yleisistä Spring Batch käytönnöistä, sovelluksessa luokat eli JavaBeanit ovat toteutettu XML-notaation sijasta Javalla omiin luokkiinsa, sillä ajossa tarvittavat prosessoinnit on haluttu esittää yhdessä luokassa usean askelluokan sijaan.

Prosessissa tarvittavat resurssit, kuten tietokantayhteydet Javax.SQL DataSource:n avulla ja Javax.Persistance Entity Manager, on määritelty kaikille ajoille yhteisessä abstraktissa kantaluokassa AMDSBatchJob. Lisäksi kantaluokka määrittelee ajon suoritukseen liittyvät parametrit ja niiden metodit, joita voidaan käyttää ajon tauottamiseen, nimen asettamiseen ja statuksen tiedusteluun. Näitä metodeja käyttää BatchController -luokka, joka sisältää rajapinnan kaikkien AMDSBatchJob-kantaluokan toteuttavien luokkien listaamiseen, käynnistämiseen ja pysäyttämiseen.

ItemReader sisältää yksinkertaisen määrityksen JdbcCursorItemReader:lle, joka ottaa parametriksi luettavan kertaosuuden (FetchSize), tietolähteen (DataSource), rivin käsittelymetodin (RowMapper) sekä SQL-lauseen, jolla tiedot haetaan tietolähteestä. Esimerkki ItemReader:n määrittelystä löytyy ohjelmasta 3.1, jossa määritetään getReader() -metodi, joka luo ja palauttaa JdbcCursorItemReader:n, jolle määritellään tietokanta, kerralla ladattavien rivien määrä, tieto tilan säilyttämisestä ja SQL-lauseen palauttava metodi. SQL-lauseen palauttava getSql() -metodi voidaan ylikirjoittaa kussakin eräajossa toteutuskohtaisesti, jolloin getReader() -metodia kutsuttaessa alustetaan ItemReader eräajon omalla SQL-lauseella automaattisesti.

Ohjelma 3.1 *ItemReader -määrittely*

```
public ItemReader<I> getReader() {
    JdbcCursorItemReader<I> reader = new
        JdbcCursorItemReader<>();
    reader.setFetchSize(1000);
    reader.setSaveState(false);
```

```
        reader.setDataSource(ds);
        reader.setSql(getSql());
        reader.setPreparedStatementSetter(this);
        reader.setRowMapper(this);
        try {
            reader.afterPropertiesSet();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return reader;
    }
```

ItemReader:lle syötteenä annettava RowMapper ottaa syötteenä tietokantarivin tuottaman ResultSetin ja muodostaa siitä järjestelmässä käytetyn olion, jonka RowMapper palauttaa. Samalla voidaan suorittaa esimerkiksi tyyppimuunnoksia ResultSetin arvoille, jotta ne saadaan tallennettua olioksi. Esimerkiksi ohjelma 3.2:ssa esitelty TecDocDriveRightLink-olion mapRow -funktio, joka luo yksinkertaisen linkkiolion tallentaen TecDocId:n, ChassisId:n ja ModelId:n.

Ohjelma 3.2 *mapRow-metodi*

```
@Override
public TecDocDriveRightLink mapRow(ResultSet rs, int rowNum)
    throws SQLException {
     TecDocDriveRightLink tdl = new TecDocDriveRightLink();
     tdl.setTecdocId(Utils.toInt(rs.getString(1)));
     tdl.setChassisId(Utils.toInt(rs.getString(2)));
     tdl.setModelId(Utils.toInt(rs.getString(3)));

     return tdl;
}
```

Esimerkissä käytetty Utils-paketti sisältää joukon apumetodeita, joita datamigraatioprosessissa käytetään esimerkiksi tyyppimuunnoksissa (Ohjelma 3.3). Tietolähteiden data ei ole täysin vakiomuotoista, joten tyyppimuunnoksissa täytyy ottaa huomioon esimerkiksi tekstimuotoiset arvot numeraalisten sijaan, joita esimerkin toInt -metodissa muutetaan tyhjiksi (null) arvoiksi. Lisäksi käytössä on MakeBatchJob:ssa tekstien normalisointi, jossa poistetaan virheelliset merkit normalizeString -metodilla

Ohjelma 3.3 *Tekstien normalisointi*

```
// Parse Integer value or return null
```

```
public static Integer toInt(String item) {
    if(StringUtils.isBlank(item) || item.equals("NULL")) {
        return null;
    }
    try {
        return Integer.parseInt(item);
    } catch (Exception e) {
        return null;
    }
}

//Normalizes string to regular letters
public static String normalizeString( String item ) {

    if( StringUtils.isBlank(item) ) {
        return item;
    }
    return Normalizer.normalize(item, Normalizer.Form.NFD)
            .replaceAll("[^\\p{ASCII}]", "");
}
```

Rivin lukemisen jälkeen mapRow-metodin muodostama olio annetaan process-metodille, joka voi edelleen käsitellä ja lopuksi palauttaa olion. Prosessoinnin jälkeen olio annetaan itemWriter -metodille, joka tallentaa objektin tietokantaan. AMDSBatchJob-kantaluokasta löytyy ItemWriter:lle toteutus, joka mahdollistaa yksinkertaisen kirjoittamisen eräajoprosessin kirjoitus-askeleessa EntityManager:n ja JPA:n avulla, kuten ohjelma 3.4:ssä on esitelty. Käytettäessä kyseistä metodia ei käyttäjän tarvitse itse toteuttaa kirjoitusmetodia, sillä Spring Batch kutsuu automaattisesti getWriter-metodia, joka on toteutettu kantaluokassa.

Ohjelma 3.4 *ItemWriter*

```
public ItemWriter<O> getWriter() {
    JpaItemWriter<O> writer = new JpaItemWriter<>();
    writer.setEntityManagerFactory(entityManagerFactory);
    return writer;
}
```

Migraatioprosessin aikana seurataan eräajojen onnistumista ja tilannetta. Jokaiselle eräajoprosessille on toteutettu getStatus, isComplete, isReadyToRun ja getCompletion -metodit. Eräajon onnistuminen palautetaan getStatus -metodilla, joka palauttaa BatchStatus -tyyppisen luetellun tyypin. Eräajon valmistuminen voidaan tie-

dustella isComplete -metodilta ja ajon aikana tilannetta pystytään seuraamaan, jos tiedetään käsiteltävien tietokantarivien kokonaismäärä, sekä sillä hetkellä käsiteltyjen rivien määrä. Tätä voidaan kysyä getCompletion -metodilta, joka palauttaa numeraalisen arvon väliltä 0-100 kertoen prosentuaalisen osuuden käsitellyistä riveistä. Koska osa eräajoista on riippuvaisia toisistaan, joudutaan tarkkailemaan myös muiden prosessien tilannetta. Tämä on toteutettu isReadyToRun -metodiin, joka palauttaa totuusarvon, jonka perusteella ajon valmius voidaan päätellä. Metodista ja siinä käytetystä isComplete -metodista löytyy esimerkki ohjelmasta 3.5, jossa tarkastellaan VTypeJob:n suoritusvalmiutta metodilla, joka käyttää alapuolella esitetyn TypeApprovalCodeJob:n isComplete -metodia. Nykyinen järjestelmä olettaa ajon olevan suoritettu, jos tilaa ei ole asetettu ja jos järjestelmästä löytyy yksikin entiteettiin liittyvä alkio.

Ohjelma 3.5 *Eräajon esivaatimukset ja niiden tarkastaminen*

```
@Override
public boolean isReadyToRun() {
    return typeApprovalCodeJob.isComplete() &&
       hsnTsnJob.isComplete() &&
       driveRightDocumentJob.isComplete()
            && tecDocDriveRightLinkJob.isComplete() &&
               upstepWheelJob.isComplete() &&
               coCVehicleJob.isComplete()
            && tpmsJob.isComplete() && makeJob.isComplete() &&
               makeDomainJob.isComplete();
}

@Override
public boolean isComplete() {
    if(status == null) {
        if (em.createQuery("FROM TypeApprovalCode t",
           TypeApprovalCode.class)
        .setMaxResults(1)
        .getResultList()
        .isEmpty())
            status = BatchStatus.UNKNOWN;
        else
            status = BatchStatus.COMPLETED;
    }
    return status == BatchStatus.COMPLETED;
}
```

Lopputuloksena eräajoista syntyy noin 43 tuhatta ajoneuvon tyyppiä kuvaavaa oliota ja niihin liittyviä varaosia noin 220 tuhatta kappaletta, jotka yhdessä muodostavat noin 69 miljoonaa linkitettyä ajoneuvo-varaosa yhteensopivuutta järjestelmän käyttöön. Prosessi toistetaan neljännesvuosittain, jotta uudet ajoneuvomallit ja vanhoihin tulleet korjaukset saadaan päivitettyä järjestelmään.

3.3 Tietomallin käyttö ajoneuvon tietomallinnuksessa

Datamigraatioprosessin lopputuloksena saatavia ajoneuvomalleja (VType) käytetään järjestelmässä esimerkiksi Trafi:n ajoneuvohaun vastauksen tietomallintamiseen. VType sisältää kuvauksen ajoneuvomallista mahdollisimman kattavine tietoineen, jolloin Trafi:n ajoneuvohaun kautta saatavaa ajoneuvoa voidaan verrata useiden eri tietokenttien pohjalta järjestelmän ajoneuvomalleihin ja päätellä näistä sopivin. Vertailua tehdään kenttä kerrallaan useisiin ajoneuvomalleihin, joista valitaan oikeellisin perustuen painotetusti yhtenevien tietokenttien lukumäärään. Painotus tarkoittaa sitä, että eri kentille annetaan eri painoarvo sen mukaan, miten hyvin ne kuvastavat ajoneuvomallia. Esimerkiksi moottorin tilavuudelle annetaan suurempi painoarvo kuin ajoneuvon massalle, sillä ajoneuvon massa saattaa riippua ajoneuvon lisävarusteista.

Taulukko 3.3 *Esimerkki ajoneuvon tietomallinnuksesta.*

Kentän tunniste	VType (VehicleType)	Trafi
Malli	ASTRA J	ASTRA
Mallityyppi	1.6	NULL
Moottorikoodi	A 16 XER	A16XER
Moottorin tilavuus	1598	1598
Moottorin teho (KW)	85	85
Sylintereiden määrä	4	4
Korimalli	HATCHBACK	CLOSED OFFROAD
Käyttövoima	PETROL	PETROL
Vetotapa	FWD	FWD
Suurin nopeus	226	188
Massa	NULL	1393
Valmistusvuosi	2009 alkaen	2010

Ajoneuvomallin tunnistamisessa käytettyjä tietokenttiä on esitelty taulukossa 3.3. Kyseisen ajoneuvon malli (VehicleType, VType) voidaan tunnistaa tietokenttiä vertailemalla, vaikka kaikki tietokentät eivät ole identtisiä sovelluksen ajoneuvomallin ja Trafi:n ajoneuvohaun vastauksen välillä. Kyseiselle mallille vahvasti painotettuja tietokenttiä ovat malli, moottorikoodi, moottorin tilavuus ja teho, sylintereiden määrä, käyttövoima ja valmistusvuosi. Esimerkiksi korimallin ja suurimman nopeuden

painoarvot ovat matalat, sillä yrityksen ajoneuvoasiantuntijat ovat huomanneet, että Trafi:n antamat arvot näiden tietokenttien osalta saattavat vaihdella paljon. Kun ajoneuvon malli on kyetty tunnistamaan, voidaan ajoneuvolle tarjota siihen sopivia varaosia, jotka ovat yhdistetty sovelluksessa ajoneuvon malliin.

3.4 Kesto ja toistettavuus

Edellä esitetty datamigraatioprosessi on toistuva. Datalähteiden tarjotessa päivityksiä, myös järjestelmän datat tulee päivittää. Toistuvuus luo vaatimuksia prosessin ajoajalle, sillä datamigraatioprosessi varaa järjestelmän resurssit käyttöönsä, jolloin järjestelmän muu samanaikainen toiminta on hidasta tai mahdotonta. Keston lisäksi toistuvuus tarkoittaa vanhan datan päivittämistä tai yliajoa, sillä jokainen datalähteen päivitys voi datan lisäämisen ja muokkaamisen lisäksi poistaa vanhaa dataa.

Taulukko 3.4 *Eräajojen kestot ja lopputulokset.*

Eräajo	Lopputulos	Kesto
DriveRightDocumentJob	DriveRightDocument	21 min
TecDocDriveRightLinkJob	TecDocDriveRightLink	7 min
VTypeJob	VType (VehicleType)	48 min
PartJob	Part	242 min
CompatiblePartJob	CompatiblePart	1080 min

Suoritusajoista (Taulukko 3.3) voidaan huomata ajoneuvon tyyppiin ja sen mallinnukseen liittyvien eräajojen olevan huomattavasti nopeampia kuin osien ja niiden yhteensopivuuksien prosessointi. Ajoneuvon mallinnus, joka koostuu DriveRightDocument -oliosta, sen VTypeen yhdistävästä TecDocDriveRightLink -oliosta ja VType -oliosta, saadaan prosessoitua järjestelmään alle 1,5 tunnissa, joka on neljännesvuosittain toistettavaksi prosessiksi inhimillinen aika. Tuloksena kolmesta eräajosta syntyvät oliot sisältävät tiedot ajoneuvomallista (VType) ja siihen ajonakaisesti yhdistetyistä rengas- ja alustatiedoista (DriveRightDocument). Yhdistämiseen käytetetään TecDocDriveRightLink -oliota, joka toimii osittain linkkitauluna, jotta eräajot voidaan toteuttaa toisistaan riippumatta. Osien ja osayhteensopivuuksien prosessointi kestää kuitenkin kohtuuttoman kauan, ottaen huomioon, että järjestelmä on sen aikaa poissa käytöstä ja sovellus on rauhoitettu pelkkien eräajojen suorittamiseen.

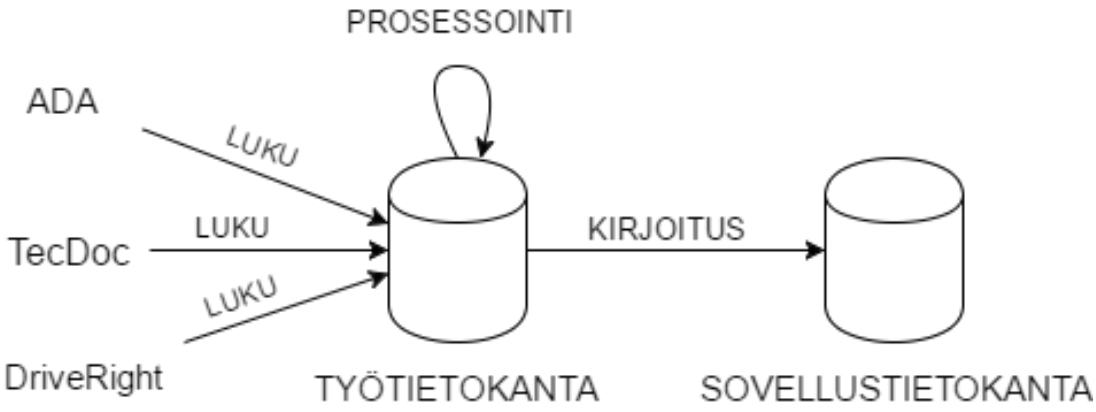
Datan päivitysprosessi on hitaampi kuin ensimmäinen datamigraatio, sillä pelkän prosessoinnin ja tietokantaan kirjoittamisen lisäksi joudutaan tietokannasta etsimään vanhoja rivejä päivitettäväksi. Päivityksen aikana Batch -moduulin resursseja voidaan rajoittaa, jotta palvelua on mahdollista käyttää, mutta vasteajat ovat huomattavasti normaalia suuremmat. Datan päivityksen sijaan voitaisiin poistaa vanha

data ja ajaa datamigraatioprosessi kuten ensimmäisellä kerralla, mutta tämä aiheuttaisi suuremman huoltokatkon prosessin ajaksi, sillä myös datan poistaminen tietokannasta vie aikaa ja datan puute aiheuttaa järjestelmän toimimattomuuden ajoneuvomallinnuksen ja varaosien tarjoamisen osalta. Koska päivitysprosessi on suoritettava vähintään neljä kertaa vuodessa, on vaihtoehtoisen prosessin etsiminen kannattavaa.

Alkuperäistä datamigraatioprosessia pyrittiin parantamaan lisäämällä päivitystä helpottavia jäsenmuuttujia olioihin, kuten aikaleimat luomiselle ja päivittämiselle. Tämän lisäksi päivitystä pyrittiin helpottamaan luomalla oliolle kevyempi poistotapa lisäämällä deleted -totuusarvo, jonka avulla jo poistettuja ajoneuvomalleja voitiin tutkia ja vertailla uusien kanssa. Nämä eivät kuitenkaan tuoneet parannuksia ite prosessin nopeuteen tai resurssien tarpeeseen.

4. DATAMIGRAATIO TYÖTIETOKANNAN AVULLA

Tavoitteena uudessa mallissa on suorittaa datamigraatio työtietokannan avulla, jotta järjestelmää kuormittava eräajo saadaan vähennettettyä minimiin. Varsinainen datamigraatioprosessi koostuu työtietokannan avulla suoritettaessa samoista, luvussa 2.1 esitetyistä, vaiheista. Data luetaan tietolähteistä työtietokantaan, käsitellään valmiiksi järjestelmän vaatimaan muotoon ja siirretään järjestelmän käyttämään tietokantaan.



Kuva 4.1 *Uudistettu datamigraatioprosessi työtietokannan avulla toteutettuna.*

Kuvassa 4.1 esitetään datamigraatioprosessi työtietokannan avulla. Prosessissa kolme tietolähdettä, TecDoc, DriveRight ja itsetuotettu data, luetaan työtietokantaan prosessointia varten. Datan prosessointi suoritetaan työtietokannassa, jonka lopputuloksena on käyttövalmis ja mahdollisimman hyvä laatuinen data sovellusta varten. Käyttövalmis data siirretään sovelluksen tietokantaan viitetaulujen avulla kopioimalla työtietokannan käsitelty data sovellustietokannan tauluihin.

4.1 Datan lukeminen työtietokantaan

Lähdedata saadaan kahdesta eri tietolähteestä, jotka ovat kuvattu tarkemmin luvussa 3.1. Tietolähteet ovat toisistaan erillisiä ja eri muotoisia, joten lukuprosesseja tarvitaan kaksi, joista jälkimmäinen, DriveRight-tietolähteen lukuprosessi, koki suuremman muutoksen. TecDoc-varaosatietokannan lukuprosessi säilyi luvun 3.1.1 mukaisena erillisellä Java-ohjelmalla suoritettavana kertaoperaationa.

DriveRight-tietolähteen lukuprosessi muuttui aiemmasta sovelluskantaan kirjoittavasta eräajoprosesssista siten, että aiempien entiteettien sijaan kirjoitetaan työtietokantaan datat ainoastaan jakaen ne omiksi tauluikseen käyttötarkoituksen perusteella. Yksittäinen eräajoprosessi lukee tietolähteen lukukannan oikeuksien rajoituksista johtuen datan kertaoperaatiolla ja kirjoittaa sen työtietokantaan samalla tarvittaessa luoden luvussa 4.2.2 esitetyt taulut. Aluksi eräajo tyhjentää työtietokannasta käytetyt taulut, jonka jälkeen kirjoittaa rivit käyttäen SQL:n INSERT -komentoa. Tämä on yksinkertaisin ja tehokkain tapa siirtää tiedot työtietokantaan, sillä lukutietokannan oikeuksista johtuen ei suora tietokannasta tietokantaan -siirto toimi.

Kun tietolähteiden datat on saatu kirjoitettua työtietokantaan, suoritetaan tietokannan nopeuttamiseksi vaadittuja tehtäviä. Näitä ovat: Tarvittavien indeksien luonti tietokantatauluille ja tietokantataulujen nopeuttaminen VACUUM ja ANALYZE -komennoilla. VACUUM -komento poistaa indeksilistauksista ylimääräiset rivit, joita on kirjoittaessa voinut syntyä. ANALYZE -komento päivittää taulujen statistiikat, joita käytetään tietokantahakujen optimoimiseen PostgreSQL:ssa.

4.2 Työtietokanta

Työtietokanta on jaettu useisiin skeemoihin versioinnin ja selkeyden vuoksi. Listaus skeemoista ja sisällöstä löytyy taulukosta 4.1. Skeemojen versiointi takaa, että myös vanhemmat tietolähteiden päivitykset pysyvät tallessa ja käyttöön saadaan aina tuorein sisältö ilman, että datan muokkausprosessissa tarvitsisi muokata SQL -tiedostoja, jotka hoitavat prosessoinnin. Tietolähteen lukemisen yhteydessä luodaan uusi skeema tietolähdepaketin versiolle, joka nimetään käyttöskeemaksi (*tecdoc_fi*, *tecdoc_ref* tai *driveright*), joista ajettavat SQL-tiedostot ottavat ja lukevat muokattavan datan.

Työtietokanta joutuu prosessoimaan suuren määrän dataa, jolloin työmuistia vaaditaan runsaasti. PostgreSQL tukee *workmem*-komentoa, jolla voidaan säätää tietokannan käytössä olevan työmuistin määrää. Työmuistia käytetään datan säilyttämiseen muistissa esimerkiksi rivien järjestämisen aikana. Toinen tietokantapalvelimen

Taulukko 4.1 *Työtietokannan skeemat.*

Skeema	Sisältö
driveright	DriverRight-tietolähteen data
tecdoc_ref	Työversio TecDoc-ajoneuvotietokannasta
tecdoc_fi	Työversio TecDoc-tuotetietokannasta
tecdoc_ref_update_0606	TecDoc-ajoneuvotietokannan versio 06.06
tecdoc_fi_update_0606	TecDoc-tuotetietokannan versio 06.06
adidas	Skeema prosessoiduille datoille ja niiden apunäkymille.

resursseja hyödyttävä asetus on *sharedbuffers*-komento, jolla asetetaan tietokannan välimuistin määrä. Suositeltava määrä on noin 1/4 osa koko palvelimen muistista, joten työtietokannassa asetus on 2GB.

4.3 Datan muokkausprosessi

Kun data on saatu luettua työtietokantaan, se muokataan vastaamaan järjestelmän tarpeita. Muokkausprosessissa tarvittavat muuttujat tyypitetään, merkkijonoista poistetaan ei-sallitut merkit ja varmistetaan, että esimerkiksi numeraaliset arvot ovat todella numeraalisia arvoja. Tietolähteiden lähdedatat ovat tarkoitettu ajoneuvomallinnuksen osalta käytettäväksi vain ihmisen luettavassa muodossa, joten niissä saattaa olla merkkejä ja arvoja, joita ei sallita koneellisesti dataa käytettäessä. Lisäksi vahva tyypitys esimerkiksi luetelluiksi tyypeiksi takaa ajoneuvon mallinnuksessa vertailukelpoisuuden järjestelmän ulkoisiin ajoneuvointegraatioihin.

4.3.1 Tyypitys

Tyypityksellä tarkoitetaan taulun sarakkeenn arvon muuttamista toiseen tyyppiin. Varsinkin DriveRight -tietolähteen lähdedata luetaan työtietokantaan merkkijonoina, sillä se sisältää esimerkiksi mittayksiköitä muuten numeraalisten arvojen joukossa. Tyypitettäessä tulee myös varmistaa, että kaikissa desimaaliluvuissa on käytetty samaa desimaalierotinta, joka on piste.

Desimaalierotin saadaan varmistettua korvaamalla käytetyt pilkut pisteillä. Korvaus onnistuu PostgreSQL:n replace-funktiolla (Ohjelma 4.1), jonka jälkeen korvausfunktion lopputulos tyyppimuunnetaan numeraaliseksi arvoksi.

Ohjelma 4.1 *replace -funktio*

```
replace(fieldname, ',', '')::numeric
```

Tyyppimuutokset luetelluksi tyypiksi vaativat apunäkymän, joka koostaa TecDoc datamallissa käytetyt numeraaliset arvot tyyppitaulun *d052* arvoista. Näkymään on otettu tyyppitaulun teksti mukaan yhdeksi sarakkeekksi, jotta näkymän oikeellisuus voidaan tarkistaa autoalan asiantuntijan toimesta, sillä kaikki luetellut arvot eivät ole selkeitä asiaan vihkiytymättömälle.

Ohjelma 4.2 *Jarrutyypin tyyppimuunnos apunäkymän avulla*

```
CREATE VIEW adidas.braketype_mappings as (SELECT key::int,
    tecdocText,
     CASE     WHEN key = '001' THEN 'HYDRAULIC'
              WHEN key = '002' THEN 'HYDRAULIC_MECHANIC'
              WHEN key = '003' THEN 'MECHANIC'
              WHEN key = '004' THEN 'COMPRESSED_AIR'
              WHEN key = '005' THEN 'HYDRAULIC'
              WHEN key = '006' THEN 'HYDRAULIC'
              WHEN key = '007' THEN 'ELECTRIC_HYDRAULIC'
              WHEN key = '008' THEN 'ELECTRIC'
              WHEN key = '009' THEN 'ELECTRIC_INERTIA'
              ELSE 'MISSING_ENUM'
         END braketype
     FROM tecdoc_ref.d052 d
     LEFT OUTER JOIN (SELECT beznr, bez as tecdocText FROM
         tecdoc_ref.d030 WHERE sprachnr = 4) c ON (d.beznr =
         c.beznr) WHERE tabnr = 84);
```

Ohjelma 4.2:n esimerkissä luodaan apunäkymä ajoneuvon jarrutyypin määrittämiseksi luetelluksi tyypiksi. Avainarvot ovat TecDoc tietolähteestä peräisin, josta ne ovat yhtenäistetty ulkoisista integraatioista ja järjestelmän tarpeista johdetuiksi arvoiksi. Itse CASE WHEN -tyyppinen arvojen määrittely vastaa yleisesti ohjelmoinnissa käytettyä Switch Case -valintarakennetta. Kyseinen näkymä tuottaa listauksen arvoista, jota voidaan käyttää TecDoc arvon tyypittämiseen järjestelmän käyttämään lueteltuun tyyppiin. Jarrutyyppien apunäkymää (Taulukko 4.2) voidaan käyttää jatkossa liittämällä apunäkymä mukaan luotavaan näkymään (Ohjelma 4.3) ja valitsemalla sieltä oikea key-saraketta vastaava arvo. Apunäkymässä käytetään normaalia näkymää materialisoidun näkymän sijaan, sillä se on kooltaan niin pieni, ettei materialisoitu näkymä tuo lukunopeuteen merkittävää eroa. Apunäkymiä tarvitaan useita, joten on käytännöllistä, että ne pysyvät ajan tasalla ilman materialisoidun näkymän REFRESH -komentoa. [10]

Ohjelma 4.3 *Jarrutyyppien liittäminen näkymään*

```
LEFT OUTER JOIN adidas.braketype_mappings b ON (b.key = bremsart)
```

Taulukko 4.2 *Apunäkymä jarrutyypin määrittelemiseksi luetelluksi tyypiksi.*

key	tecdocText	braketype
1	Hydraulic	HYDRAULIC
2	Hydraulic-mechanical	HYDRAULIC_MECHANIC
3	Mechanical	MECHANIC
4	Compressed Air	COMPRESSED_AIR
5	Hydraulic without brake booster	HYDRAULIC
6	Hydraulic with brake booster	HYDRAULIC
7	Electro-hydraulic	ELECTRIC_HYDRAULIC
8	Electronic	ELECTRIC
9	Electromechanical	ELECTRIC_INERTIA

Lueteltujen tyyppien lisäksi apunäkymää käytetään totuusarvojen muutoksissa. TecDoc käyttää totuusarvona numeroita 0, 1, 2 ja 9. Tietolähteen dokumentaation perusteella nämä voidaan muuttaa totuusarvoiksi apunäkymän avulla (Ohjelma 4.4), jota käytetään samaan tapaan kuin jarrutyyppien apunäkymää (Ohjelma 4.3). Koska totuusarvoja löytyy vain taulusta tecdoc_ref.d120, voidaan valita näkymään vain taulussa esiintyvät arvot. Tällöin saadaan näkymä, jonka sisältö on esitetty Taulussa 4.3.

Ohjelma 4.4 *Totuusarvojen apunäkymä*

```
CREATE VIEW adidas.boolean_mappings as (SELECT distinct on(abs)
    abs as key,
     CASE WHEN abs = 1 THEN TRUE
          WHEN abs = 0 THEN FALSE
          WHEN abs = 2 THEN FALSE
          WHEN abs = 9 THEN NULL
       END booleanvalue
       FROM tecdoc_ref.d120);
```

Taulukko 4.3 *Apunäkymä totuusarvojen määrittelemiseksi.*

key	booleanvalue
0	f
1	t
2	f
9	[NULL]

4.3.2 Merkkijonomuutokset

Merkkijonomuutoksia voidaan käyttää merkkijonojen (text, varchar) muokkaamiseen. Muunnokset perustuvat PostgreSQL:n merkkijonofunktioiden käyttöön tietokantahakujen yhteydessä. Funktioilla voidaan esimerkiksi jakaa pilkulla eroteltu

merkkijonolista JSON-muotoiseksi listaksi tai useammaksi riviksi.

Taulukko 4.4 *Esimerkkejä merkkijonomuutoksista.*

arvo	funktio	lopputulos
API SM, API SL	regexp_split_to_array(arvo, ',')	'API SM', 'API SL'
API SM, API SL	regexp_split_to_table(arvo, ',')	API SL API SM
110 kW	substring(arvo from '([0-9]1,4)')	110

Taulukossa 4.4 on esitelty kolme työssä käytettyä merkkijonomuutosta. Esimerkeissä olevat arvot ovat realistisia, työssä esiintyviä muunnoksia. API SM ja API SL ovat API-standardin mukaisia öljyluokituksia, jotka ovat luettu tietokantaan pilkulla eroteltuna listana yhdessä merkkijonossa. Jotta arvoja voidaan käyttää kohdistamaan moottoriöljyjä kyseiseen ajoneuvomalliin, tulee kukin öljyluokka erottaa omaksi merkkijonokseen. Tähän voidaan käyttää kahta eri säännölliseen lauseeseen (Regular Expression, regexp) perustuvaa jakofunktiota, joista ensimmäisenä esitetty *regexp_split_to_array()* jakaa annetun arvon JSON-muotoiseen listaan ja *regexp_split_to_table()* jakaa annetun arvon useaksi riviksi. Taulukon viimeinen esimerkkifunktio ottaa syötteeksi tehollisarvoa kuvaavan merkkijonon "110 kW"ja poistaa siitä mittayksikön suodattaen syötteestä säännöllisen lausekkeen avulla kaikki muut merkit lukuunottamatta numeroita 0-9. Tällöin merkkijono voitaisiin tallentaa numeraaliseen muotoon merkkijonon sijaan.

Merkkijonomuutoksia käytetään hakulauseissa. Tällöin lähdetaulusta voidaan ottaa suoraan näkymään data halutussa muodossa, jolloin aina näkymää käytettäessä se muodostaa oikeellisen datan. Edellä mainituista merkkijonofunktioista vain *substring()* kuuluu SQL-määrittelyn vakiofunktioihin. Merkkijonon jakofunktiot ovat PostgreSQL:n omia funktioita, joka tarjoavat käyttäjälle helpomman syntaksin ja toiminnallisuuden, mutta sisäisesti ne perustuvat SQL-määrittelyn vakiofunktioihin. [10]

4.4 Datan yhdistäminen

Lähdedata sijaitsee useissa eri skeemoissa ja tauluissa työtietokannassa. Jotta siirto järjestelmän tietokantaan on mahdollisimman yksinkertainen, data tulee saada yhtenäiseen, järjestelmän vaatimaan, muotoon. Yhdistäminen tapahtuu luomalla materialisoituja näkymiä, jotka hyödyntävät edellä käsiteltyjä datan muokkaus- ja tyypityskeinoja. Materialisoituja näkymiä luodaan yksi kutakin järjestelmän migroitavaa tietokantataulua kohden. Näkymät luodaan työtietokannan *adidas* -skeemaan, jotta ne pysyvät selkeästi erillään.

Ohjelma 4.5 *CompatiblePart -näkymä*

```
CREATE MATERIALIZED VIEW adidas.pc_compatible AS
    SELECT row_number() OVER () AS id,
    a.id AS part_id,
    tab.genartnr AS genericpart_id,
    tab.lfdnr,
    tab.ktypnr
   FROM adidas.part a
     JOIN (
         SELECT d400.artnr,
             d400.dlnr,
             d400.genartnr,
             d400.lfdnr,
             d400.kritwert::integer AS ktypnr
           FROM adidas.d400_custom
          WHERE d400.kritnr = 2
        UNION
         SELECT d400.artnr,
             d400.dlnr,
             d400.genartnr,
             d400.lfdnr,
             d400.vknzielnr AS ktypnr
           FROM tecdoc_work.d400
          WHERE d400.vknzielart = 2) tab ON (a.articleNumber =
              tab.artnr AND a.supplierId = tab.dlnr);
```

Materialisoidut näkymät yhdistelevät tietoa eri tauluista, luovat tarvittavia tunnisteita ja tekevät edellä esitettyjä tyyppimuunnoksia. Esimerkkiohjelmassa (Ohjelma 4.5) luodaan materialisoitu näkymä osien yhteensopivuudelle. Se sisältää näkymään rivinumerosta generoidun tunnisteen (id), geneerisen osanumeron (genericpart_id), osan tunnistenumeron (part_id) ja ajoneuvomallin tunnistenumeron (ktypnr). Tiedot koostetaan osa-näkymästä (adidas.part), lisätaulusta (adidas.d400_custom) ja yhteensopivuustaulusta (tecdoc_work.d400). Lisätaulu (adidas.d400_custom) sisältää TecDoc:n ulkopuolista osayhteensopivuustietoa, jota suuremmat varaosatoimittajat tuottavat omille varaosilleen. Se on täysin yhteensopivaa TecDoc-varaosadatan kanssa, mutta sitä säilytetään työtietokannassa omassa taulussaan, jotta sitä voidaan päivittää eri tahtiin muun varaosadatan kanssa.

4.4.1 Tietolähteiden priorisointi

Tietolähteet tarjoavat osittain päälleikkäistä tietoa ajoneuvomalleista. Tietolähteiden lisäksi yrityksen alan asiantuntijat tuottavat tarvittaessa ajoneuvon mallinnuksen avuksi dataa, joka yhdistetään priorisointiprosessissa muiden tietolähteiden tuottamaan dataan. Datalähteiden priorisointi perustuu manuaalisesti datan tutkimiseen, jotta voidaan päättää luotettavin datalähde kullekkin tietokantataululle. Järjestelmän kompleksisuutta vähentääkseen duplikaattidataa ei oteta mukaan järjestelmään ajettavaan dataan, vaan data priorisoidaan työtietokannassa, jolloin voidaan tuoda järjestelmän tietokantaan parasta mahdollista tietoa.

Tietolähteiden priorisointia varten tulee samaan olioon liittyvät datat olla yhdistettävissä. Tämä onnistuu ajoneuvomallien osalta DriveRight -tietolähteen tarjoaman Ktype chassisId modelId -taulun avulla, joka yhdistää TecDoc:n ajoneuvomallin tunnisteen (Ktype) DriveRight korimallin (chassisId) ja ajoneuvomallin (modelID) tunnisteisiin. Näillä tunnisteilla voidaan yhdistää kahden tietokantataulun kukin rivi toisiinsa ja verrata niitä keskenään. Vertailua varten voidaan luoda apunäkymiä, jotka yhdistävät edellä kuvatulla tavalla kaikkien tietolähteiden taulut. Apunäkymä voidaan järjestää TecDoc tunnisteen mukaisesti, jotta allekkain ovat eri datalähteiden tiedot. Priorisoinnissa kullekkin raakadatataululle määritetään prioriteettiarvo asteikolla 1-3, jossa 1 on korkein prioriteetti ja 3 matalin.

4.4.2 Datan kerrosmalli

Apunäkymää, jossa on allekkain eri datalähteiden rivit järjestettynä prioriteetin mukaan kutsutaan *kerrosmalliksi*. Kerrosmalli sai projektissa lempinimen *adidas*-malli, sillä alkuun se sisälsi dataa kolmessa eri kerroksessa urheilumerkin logoa mukaillen. Kerrosmallissa voidaan ajatella, että yhteen olioon liittyvä data on kerroksissa sen mukaan, mistä tietolähteestä se on peräisin. Kerrostaminen on viimeinen välivaihe ennen datan muokkaamista lopulliseen materialisoituun näkymään, josta se siirretään järjestelmän tietokantaan. Sen tarkoitus on helpottaa eri tietolähteiden datan laadun hahmottamista priorisoimisen myötä, sekä mahdollistaa olion tietojen litistäminen yhdeksi tietoriviksi prioriteettiarvon mukaisesti. Litistämisessä korkeimman prioriteetin omaava rivi ylikirjoittaa matalamman prioriteetin tietorivit niiden kolumneiden osalta, joissa sillä on arvoja asetettuna. Toisena prioriteettijärjestyksessä oleva rivi täyttää tyhjät arvot ja lopuksi kolmas rivi täydentää vielä edelleen tyhjät arvot. Lopputulos on paras ja kattavin tietorivi oliota varten.

Litistämisprosessi yhden olion datan osalta on esitetty taulukossa 4.5. Siinä olion tietorivi lopulliseen työnäkymään saadaan keräämällä kolmesta toisiinsa TecDoc tun-

Taulukko 4.5 *Kerrosmallin litistäminen yhteen riviin prioriteettijärjestyksessä.*

prio	ktypnr	chassisid	modelid	drivetype	braketype	brakesystem	weight
1	14	NULL	NULL	FWD	NULL	NULL	NULL
2	14	32	168	AWD	NULL	NULL	990
3	14	NULL	NULL	NULL	HYDRAULIC	DISC	NULL
	14	**32**	**168**	**FWD**	**HYDRAULIC**	**DISC**	**990**

nisteella (ktypnr) yhdistetystä rivistä. Korkeimman prioriteetin rivistä (prio arvolla 1) valitaan kentät ktypnr ja drivetype. Tämä rivi kuvaa yrityksessä itse tuotetettua ajoneuvomallinnusdataa, jolla voidaan korjata ulkoisten tietolähteiden virheitä. Koska muut arvot riviltä ovat tyhjiä (NULL), ne eivät ylikirjoita alemman prioriteetin rivien arvoja. Toiselta riviltä (prio arvolla 2) data on otettu DriveRight -tietolähteestä, jossa on mukana chassisid ja modelid -tunnisteet, drivetype ja weight, joista ainoastaan drivetype:lle löytyy arvo korkeamman prioriteetin omaavalta riviltä, joten sitä ei tältä riviltä oteta huomioon. Arvot chassisid, modelid ja weight päätyvät merkitseviksi arvoiksi litistysprosessin myötä. Kolmannen eli alimman prioriteettinumeron rivillä asetettuja arvoja ovat braketype ja brakesystem, joista kumpaakaan ei korkeamman prioriteetin riveiltä löydy, joten ne päätyvät lopulliseen riviin arvoiksi. Litistysprosessin lopputuloksena esimerkistä löytyy rivi, joka sisältää parasta mahdollista dataa oliota varten, kerättynä kolmesta eri tietolähteestä.

Ohjelma 4.6 *Litistämisprosessi COALESCE -funktion avulla*

```
CREATE MATERIALIZED VIEW adidas.axle AS (SELECT
  COALESCE(a.ktypnr, b.ktypnr, c.ktypnr) AS ktypnr,
  COALESCE(a.chassisid, b.chassisid, c.chassisid) AS chassisid,
  COALESCE(a.modelid, b.modelid, c.modelid) AS modelid,
  COALESCE(a.drivetype, b.drivetype, c.drivetype) AS drivetype,
  COALESCE(a.braketype, b.braketype, c.braketype) AS braketype,
  COALESCE(a.brakesystem, b.brakesystem, c.brakesystem) AS
      brakesystem,
  COALESCE(a.weight, b.weight, c.weight) AS weight
FROM
  (SELECT * FROM adidas.axle_helper WHERE prio = 1) a,
  (SELECT * FROM adidas.axle_helper WHERE prio = 2) b,
  (SELECT * FROM adidas.axle_helper WHERE prio = 3) c);
```

Litistämisprosessissa käytetään hyödyksi PostgreSQL:n *COALESCE* -funktiota, joka palauttaa listasta ensimmäisen arvon, joka ei ole tyhjä (NULL). Litistämistä varten jokainen kolmen rivin osio täytyy valita sarakkeittain listamuotoon, jotka on järjestetty prioriteetin mukaisesti. Näistä listoista *COALESCE* -funktio osaa valita prioriteetiltaan korkeimman arvon, joka ei ole tyhjä. Esimerkkiohjelmassa 4.6 luo-

daan materialisoitu näkymä työnäkymän adidas.axle_work -kerrosmallisista riveistä, joissa on kolmea eri prioriteettiä. Jokainen prioriteetti liitetään mukaan erikseen ja nimetään arvoilla a, b ja c. Näistä muodostetaan *COALESCE* -funktiota varten listat, joissa ovat saman sarakkeen arvot jokaisesta prioriteettiluokasta. Funktio valitsee ensimmäisen arvon, joka ei ole tyhjä ja arvo nimetään *AS* -merkinnän avulla. Lopputuloksena on materialisoitu näkymä, joka sisältää saralleet ktypnr, chassisid, modelid, drivetype, braketype, brakesystem ja weight. Näkymän sisältö on litistetty data parasta mahdollista tietoa, valmiina siirrettäväksi järjestelmän tietokantaan.

4.5 Siirto tuotantojärjestelmään

Tuotantojärjestelmään siirto koostuu useasta osasta. Alkuvalmisteluina täytyy luoda yhteys tuotantojärjestelmän tietokannan ja työtietokannan välille, mikä sisältää asetusten määrittelyn ja käyttäjäoikeuksien asettamisen. Työtietokannan näkymät yhdistetään järjestelmän tietokantaan vierastauluilla, jonka jälkeen järjestelmän tietokannan data voidaan korvata työtietokannasta valituilla datoilla. Siirtoprosessiin kuuluu myös siirron jälkeinen optimointi, joka sisältää tietokantataulujen indeksoinnin sekä tarvittavat siivous- ja analysointitoimenpiteet tietokannan nopeuttamiseksi.

4.5.1 Työtietokannan ja järjestelmätietokannan yhteys

Yhteys kahden PostgreSQL tietokannan välille muodostetaan luvussa 2.2.2 kuvatulla vierastietokantayhteydellä. Yhteyttä varten järjestelmän tietokantaan asennetaan *postgres_fdw* -laajennus, joka sisältää tuen ulkopuolisen tietokantayhteyteen ja tietojen sitomiseen. Laajennuksia PostgreSQL -tietokantoihin asennetaan *CREATE EXTENSION* -komennolla, joka vaatii käyttäjältä ylläpitäjän oikeuksia (superuser tai database owner) tietokantaan. [10] Yhteys toiseen tietokantaan luodaan *CREATE SERVER* -komennolla, jolle annetaan parametrinä lisätty laajennus (Ohjelma 4.7).

Ohjelma 4.7 *Tietokantayhteyden luominen*

```
CREATE SERVER staging FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (host 'localhost', port '5432', dbname 'staging');
```

Yhteyden määrittämisen jälkeen vaaditaan tietokantojen käyttäjätunnusten sitominen toisiinsa. Tämä takaa sen, että järjestelmän tietokantakäyttäjästä osataan päätellä oikea työtietokannankäyttäjä ja autentikoida oikealla salasanalla. Käyttäjien

yhdistäminen tapahtuu *CREATE USER MAPPING* -komennolla, jolle parametrinä annetaan järjestelmätietokannan käyttäjä, työtietokannan nimi, joka on aiemmin määritetty tietokantayhteyttä luodessa, sekä työtietokannan käyttäjätunnukset (Ohjelma 4.8).

Ohjelma 4.8 *Tietokantakäyttäjien sitominen toisiinsa*

```
CREATE USER MAPPING FOR dbuser SERVER staging
  OPTIONS (user 'adauser', password 'salasana');
```

4.5.2 Vierastaulut

Vierastaulut muodostetaan tuotantojärjestelmän tietokantaan käyttämällä PostgreSQL:n foreign data wrapper -laajennusta (postgres_fdw), joka sitoo tietokannan vierastaulun työtietokannan materialisoituun näkymään. Järjestelmän tietokanta ei tiedä millainen työtietokannan materialisoidun näkymän rakenne on, joten vierastaulun rakenne pitää määrittää sarake kohtaisesti vierastaulua luodessa. Vierastaulu luodaan *CREATE FOREIGN TABLE* -komennolla, jolle annetaan parametrinä vierastietokannan nimi ja taulun tiedot, sisältäen skeeman ja nimen (Ohjelma 4.9).

Ohjelma 4.9 *Esimerkki vierastaulun luomisesta*

```
CREATE FOREIGN TABLE foreign_compatiblepart (
    id BIGINT,
    part_id BIGINT,
    genericpart_id INTEGER,
    ktypnr INTEGER
) SERVER staging
  OPTIONS (schema_name 'adidas', table_name 'pc_compatible');
```

Esimerkissä (Ohjelma 4.9) luodaan vierastaulu osayhteensopivuudelle. Huomattavaa on, että parametrinä annettu *table_name* voi osoittaa myös materialisoituun näkymään. Vierastaulua voidaan käyttää kuten normaalia tietokantataulua hakuoperaatioissa. Operaatioiden nopeuteen vaikuttaa suuresti tietokantojen välisen yhteyden nopeus, sillä vierastaulu on toisen tietokannan taulu, joka on sidottu tietokantaan hakuoperaatioita varten. Tietokantahaut siirtyvät foreign data wrapper -laajennuksen avulla suoritettaviksi työtietokantaan, joka toteuttaa haun ja palauttaa hakutuloksen. Vierastaulut eivät tue kirjoitus-, päivitys- tai poisto-operaatioita.

4.5.3 Tuotantojärjestelmän päivitys

Tuotantojärjestelmän datapäivitys sisältää kaksi kokonaisuutta. Ensiksi päivitetään ajoneuvomallien datat, jonka jälkeen päivitetään tuotteet ja niihin liittyvät tiedot, sekä ajoneuvomallien ja tuotteiden osayhteensopivuudet. Päivitykset eroavat toisistaan sillä, että ajoneuvomallien dataa ei voida poistaa suoraan, sillä niihin perustuvat useat yhteydet, kuten riippuvuudet ajoneuvojen huoltoihin, varsinaisiin ajoneuvoihin ja muihin ajoneuvomallinnuksesta riippuvaan toiminnallisuuteen katoaisivat päivityksessä. Asia on ratkaisu käyttämällä ajoneuvomalleissa poistettu -parametriä (deleted), jonka avulla järjestelmästä poistuneet ajoneuvomallit säilyvät tietokannassa ja ovat käytettävissä uusia linkityksiä luotaessa. Tuotteisiin ja niiden osasopivuuksiin liittyvät taulut voidaan poistaa ja luoda uusiksi vierastaulujen pohjalta.

Ajoneuvomallien päivitysprosessin ensimmäinen askel on merkata poistuneet ajoneuvomallit poistettu -parametrillä. Se onnistuu ajamalla *UPDATE* -komento ajoneuvomallien taulun niille riveille, joita ei löydy ajoneuvomallien vierastaulusta. Komennolla asetaan deleted-sarakkeen arvoksi tosi ja päivitysajaksi nykyhetki, joka saadaan *NOW()* -funktiolla (Ohjelma 4.10). Päivitettyjen ajoneuvomallien lisäyksessä hyödynnetään moottoritiedot sisältävää vtype_motor -taulua, josta löytyvät ajoneuvomallin yksilöivät tunnisteet (tecdocid ja motorid). Itse ajoneuvomallin järjestelmän tunniste generoidaan tietokannan sekvenssistä, jota myös järjestelmän Hibernate käyttää.

Ohjelma 4.10 *Ajoneuvomallien poistaminen ennen uuden datan lisäämistä.*

```
UPDATE vtype SET deleted = true, updated = NOW()
  WHERE tecdocid NOT IN (SELECT tecdocid FROM foreign_modelinfo);
INSERT INTO vtype (id, tecdocid, motorid)
  SELECT nextval('hibernate_sequence'), tecdocid, motorid
  FROM vtype_motor
  WHERE motorid NOT IN (SELECT motorid FROM vtype);
```

VType-entiteetti on varsinaisesti vain ajoneuvomallin osia sitova, joten sen tiedot eivät päivity. Se kuitenkin liittää ajoneuvomallin muut osat yhteen, joten sitä voidaan käyttää merkkaamaan, onko kyseinen ajoneuvomallin olio poistunut, jolloin siihen liittyvät taulut voidaan poistaa ja luoda uudestaan päivitetyillä tiedoilla suoraan vierastauluista (Ohjelma 4.11).

Ohjelma 4.11 *Ajoneuvomallinnuksen akseli ja alustatiedon päivitys.*

```
DELETE FROM vtype_axles;
INSERT INTO vtype_axles SELECT * FROM foreign_axles;
```

Kuten ajoneuvomallin osataulut, myös tuotteet, tuotetiedot ja osasopivuudet, voidaan päivittää yksinkertaisesti poistamalla vanhat rivit ja lisäämällä uudet vierastaulun kautta. Tämä nopeuttaa päivitysprosessia huomattavasti, sillä tietokanta pystyy taulun poiston yhteydessä vapauttamaan siihen sidotut resurssit eivätkä esimerkiksi taulun indeksit hidasta tauluun kirjoittamista (Ohjelma 4.12).

Ohjelma 4.12 *Osayhteensopivuustaulun päivittäminen*

```
DROP TABLE IF EXISTS compatiblepart CASCADE;
CREATE TABLE compatiblepart
  AS SELECT * FROM foreign_compatiblepart;
```

Komennossa (Ohjelma 4.12) mukana oleva *CASCADE* -parametri poistaa kaikki tauluun vierasavaimilla liittyvät taulut, jolloin niitä ei tarvitse poistaa erikseen. Ennen järjestelmän uudelleenkäyttöönottoa tauluille täytyy luoda indeksit ja taulu analysoida, jotta tietokanta osaa optimoida niihin tehtävät haut. Indeksit luodaan käyttäen fillfactor -parametriä, joka kertoo tietokannalle kuinka täyteen indeksien hakusivut voidaan täyttää. Staattisille tauluille suositellaan täyttä fillfactor-arvoa eli 100, jolloin indekseille varataan muistia vain sen verran, kuin taulun koosta voidaan sillä hetkellä päätellä. Jos taulun koko vaihtelee suuresti käytössä käytetään pienempää arvoa. Koska järjestelmään lisätään tuote- ja ajoneuvomallidataa vain päivitysprosessin myötä, voidaan käyttää indekseissä arvoa 100. Indeksi luodaan taulun jokaiselle sarakkeelle, jota käytetään jossain tietokantahaussa. Indeksien luomisen jälkeen ajetaan komento *VACUUM ANALYZE*. *VACUUM* on PostgreSQL:n roskankerääjä, joka vapauttaa tauluille varattua tilaa tietokannasta. Se on yleensä tietokannassa taustalla pyörivä prosessi, mutta koska päivityksessä tuodaan miljoonia uusia rivejä tietokantaan, ajetaan siivousprosessi manuaalisesti, jotta saadaan suorituskykyä nostettua heti päivityksen yhteydessä, sekä tällöin voidaan itse vaikuttaa milloin aikaa vievä siivousprosessi suoritetaan. *ANALYZE* -parametri komennon yhteydessä analysoi taulua ja sen indeksejä, jotta tietokanta pystyy suorittamaan siihen hakuja mahdollisimman tehokkaasti (Ohjelma 4.13).

Ohjelma 4.13 *Indeksien luominen osayhteensopivuustauluun*

```
CREATE INDEX ON compatiblepart(part_id) WITH (fillfactor=100);
CREATE INDEX ON compatiblepart(genericpart_id) WITH
    (fillfactor=100);
CREATE INDEX ON compatiblepart(ktypnr) WITH (fillfactor=100);
VACUUM ANALYZE compatiblepart;
```

Sovelluspohjainen datamigraatioprosessi uudistettiin käyttäen apuna työtietokantaa, jonka avulla migraatiprosessin datan muokkaus -vaihe pystyttiin suorittamaan

tietokannassa. Datan muokkaus koostui näkymillä ja merkkijonomuutoksilla toteutettuihin muutoksiin, joilla dataa tyypitettiin ja jäsenneltiin. Tietolähteiden priorisointiin datan yhdistämisessä kehitettiin kerrosmalli, jolla data muokattiin olioille sopivaan muotoon. Valmis data pystyttiin siirtämään sovelluksen tietokantaan luomalla sovelluksen tietokannan ja työtietokannan välille vierastietokantayhteys, jonka avulla sovelluksen tietokantaan luotiin työtietokannan tauluja vastaavat viitetaulut. Viitetauluista tiedot pystyttiin kopioimaan sovelluksen tietokantatauluihin käyttäen hyväksi pelkästään tietokannan ominaisuuksia. Siirron jälkeen sovellustietokantaan suoritettiin suorituskykyä parantavia toimia, kuten indeksien luonti ja analysointi.

5. PROSESSIEN ARVIOINTI

Datamigraatioprosessin uudistamisessa tavoitteena oli päästä eroon aikaa ja resursseja vievistä eräajoista, joilla järjestelmän tärkeimmän datamallit pidettiin ajantasalla. Päivitykset ovat osa jatkuvaa järjestelmän ylläpitoa, joten niiden luotettavuus ja helppous olivat myös osana tavoitteita, jotka uudistusprojektille asetettiin. Projektille määritetyt tavoitteet olivat:

- Luopua mahdollisimman suurelta osin eräajoista
- Ylläpitää jäljitettävyys tietolähteisiin
- Optimoida prosessin kesto ja resurssien kulutus
- Pyrkiä luomaan helposti toistettava prosessi
- Dokumentoida prosessi, jotta kuka tahansa yrityksen ohjelmistokehittäjistä pystyy siihen

Uudistetun prosessin resurssien kulutus ja kesto mitattiin vaiheittain päivitysprosessin aikana, sillä työtietokannassa suoritettavat vaiheet ajettiin yhtenä SQL-tiedostona tietokantaan, jonka kesto mitattiin. Samoin meneteltiin itse järjestelmädatojen päivityksessä järjestelmän tietokantaan. Arvion lähtökohdaksi aiempi datamigraatiototeutus on kuvattu luvussa 3.

5.1 Resurssien kulutus ja kesto

Suurin muutos aiempaan datamigraatioprosessiin on, että uudessa prosessissa datan muokkaus ja käsittely tapahtuu työtietokannassa, jolloin se ei vie resursseja sovelluspalvelimelta tai sovellustietokannasta. Uudessa prosessissa ainoastaan DriveRight -tietolähteen lukeminen lukutietokannasta työtietokantaan hoidetaan järjestelmän eräajoprosessilla, joka vie aikaa vain 18 minuuttia. Järjestelmän tietokantaa tämä osa migraatioprosessia ei rasita.

Taulukko 5.1 *Uudistetun prosessin kestot verrattuna aiempaan.*

Korvattu eräajo	Luku	Prosessointi	Siirto	Yhteensä	Aiempi prosessi
DriveRightDocumentJob	18 min	9 min	-	27 min	21 min
TecDocDriveRightLinkJob	-	-	-	-	7 min
VTypeJob	-	12 min	14 min	26 min	48 min
PartJob	-	77 min	28 min	105 min	242 min
CompatiblePartJob	-	110 min	92 min	202 min	1080 min

Taulukossa 5.1 on esitetty uuden prosessin kesto vaiheittain ja vertailukohta aiemmasta prosessista. Kolumnissa Luku on esitetty tietolähteiden siirto työtietokantaan. Aiemmista eräajoista se on laskettu kokonaisaikaan vain DriveRightDocumentJob ja TecDocDriveRightLinkJob -ajoissa, jotka ovat yhdistetty uudessa migraatioprosessissa, joten DriveRightDocumentJob sisältää myös aiemman TecDocDriveRightLinkJob -prosessin. Prosessointi taulukossa tarkoittaa työtietokannassa muodostettavia näkymiä ja aikaa, joka kuluu tietokannan suorittaessa ajettavaa SQL-tiedostoa, joka luo apunäkymät ja lopulliset materialisoidut näkymät adidas-skeemaan. Siirto tarkoittaa datan siirtämistä työtietokannasta järjestelmän tietokantaan luvun 4.5.3 mukaisesti. Siirron hitain osa on tietojen kirjoittaminen työkantaan, sillä Amazonin RDS -tietokannat rajoittavat kirjoitusnopeutta IOPS:n mukaisesti. Yhteensä -kolumnin arvoihin ei ole laskettu mukaan lukuvaiheen aikaa käsiteltäessä VTypeJob, PartJob ja CompatiblePartJob -osioita, sillä niiden aiemman prosessin mittauksissa ei kyseisiä aikoja huomioitu.

Datamigraatioprosessin jokaisen osa-alueen kestot lyhenivät uudistuksen myötä. Suurimman muutoksen kokivat raskaimmat osiin ja osayhteensopivuuksiin liittyvät ajot, jotka ovat prosessin suurimmat myös rivimääriltään. Näiden suhteen prosessiuudistus lyhensi ajo aikaa parhaimmillaan yli 14 tuntia. Huomattavaa on, että aiemmassa datamigraatiossa järjestelmä olisi ollut koko tämän ajan lähes toimintakyvytön vastaamaan käyttäjien kutsuihin. Nykyisessä prosessissa järjestelmän huoltokatko datan siirtovaiheessa on mittausten perusteella 134 minuuttia, joka on neljästi vuodessa ajettavasta operaatiosta siedettävä aika.

Uudistettu prosessi säästää järjestelmän resursseja, sillä datan prosessointi tapahtuu työtietokannassa järjestelmän sijaan. Työtietokannan resurssit ovat helposti jatkossa kasvatettavissa, sillä pilvipohjaisessa RDS -tietokannassa resurssejen skaalaus on helppoa ja se voidaan tehdä tarvittaessa. Jatkossa on mahdollista lisätä työtietokannan resursseja datan prosessoinnin ja siirron ajaksi prosessin nopeuttamiseksi. Koska varsinainen sovellus on tuotantokäytössä, tulee sen resurssit olla pääsääntöisesti sen keskeisimpiin toimintoihin varattuina, eikä järjestelmän datapäivitysprosessia lasketa niihin. Tällöin datamigraatioprosessin erottaminen sovellusjärjestelmästä palvelee

sovelluksen käyttäjiä.

5.2 Toistettavuus ja ylläpidettävyys

Datamigraatioprosessi ajetaan tällä hetkellä neljä kertaa vuodessa tietolähteiden päivitysten takia. Tulevaisuudessa TecDoc on suunnitellut tihentävänsä julkaisutahtia, jolloin päivitysprosessi suoritettaisiin kerran kuukaudessa. Tämä luo vaatimukset prosessin toistettavuudelle, sillä päivitykseen kulutettu aika kertaantuu vuodessa useaksi työpäiväksi. Toistettavuuteen vaikuttavat myös datamigraatioprosessin luotettavuus, sillä virheiden todennäköisyys kasvaa, kun prosessi joudutaan suorittamaan useammin.

Uudistuksessa ei päästy vielä täysin eroon manuaalisesta työstä. Tietolähteiden datat tulee lukea työtietokantaan ohjelmallisesti ja sekä työ- että järjestelmätietokantoihin ajaa yhteensä kolme SQL -tiedostoa, jotka hoitavat datan prosessoinnin sekä siirron työtietokannasta järjestelmän tietokantaan. Varmuuden vuoksi sovellus suljetaan siirron ajaksi manuaalisesti pysäyttämällä sovellusta tarjoava Tomcat -ohjelma. Näiden vaiheiden lisäksi tietolähteiden datat tarkastetaan pintapuolisesti työtietokannassa ennen prosessointia, jotta voidaan varmistua tietojen onnistuneesta lukemisesta työtietokantaan.

Aiemmassa datamigraatioprosessissa TecDoc-tietolähteen lukuprosessi oli vastaava kuin uudessa prosessissa. Tämän lisäksi manuaalista työtä oli eräajoprosessien käynnistäminen hallintapaneelista oikeassa järjestyksessä. Koska aiemman prosessin eräajot veivät useita tunteja aikaa, tuli prosessia suorittavan henkilön odottaa aiempien, esiehtoina olleiden, eräajojen päättyminen ennen seuraavan käynnistämistä. Usein tämä vaati yhdeltä henkilöltä osittaisen työpanoksen viikonlopun ajalta, sillä prosessi hidasti järjestelmää niin suuresti, ettei sitä pystytty ajamaan viikon aikana, kun järjestelmällä oli kovin käyttöaste. Lisäksi sovelluksessa tehty datan prosessointi oli virheherkempää, sillä vaikka Spring Batch -kehyksellä toteutetut eräajot osaavat säilyttää tilansa useimmissa häiriötapauksissa, esimerkiksi sovelluspalvelimen kaatuminen olisi katkaissut prosessin ja jättänyt sen tilaan, josta ei olisi pystytty palautumaan ilman kyseisen eräajon aloittamista alusta.

Datamigraatioprosessin päivittämisessä toistettavuus parani huomattavasti, sillä prosessin keston lyhentyminen vähensi prosessin vaatimaa työaikaa. Järjestelmän lähes vuorokauden pituinen hidastuminen korvaantui alle kahden tunnin järjestelmäkatkolla. Manuaalisesta työstä ei vielä tässä projektissa päästy täysin eroon, mutta sen määrä väheni. Datan prosessoinnin siirtyminen työtietokantaan toi mukaan toimintavarmuutta, sillä tietokanta on varmatoiminen, eikä mahdolliset häiriötilan-

teet haittaa itse järjestelmän toimintaa prosessointivaiheessa. Lisäksi prosessoinnin nopeutuminen ja prosessoinnin suorittaminen yhdellä ajettavalla SQL-tiedostolla vähentää häiriötilanteista johtuvien uudelleen prosessointien kuormitusta prosessia suorittavaan henkilöön, sillä SQL-tiedoston voi ajaa uudelleen vain yhdellä komennolla. Koska data siirretään työtietokannasta järjestelmän tietokantaan transaktion sisällä ja yksi taulu kerrallaan, yksikin virhe prosessissa palauttaa tietokannan tilan prosessia edeltävään tilaan, eikä aiheuta muita ongelmia.

Tulevaisuudessa tietolähteiden dataformaatti saattaa muuttua tai uusia tietolähteitä voidaan ottaa mukaan järjestelmään. Tällöin datamigraatioprosessin täytyy olla muokattavissa myös uuteen dataformaattiin tai laajennettavissa uusiin tietolähteisiin. Aiemmassa prosessissa uusi tietolähde olisi lisännyt suoritettavan prosessoinnin määrää sovelluksessa ja aiheuttanut ohjelmallisia muutoksia järjestelmän eräajoihin. Uudistetussa prosessissa muutokset kohdistuvat isolta osin työtietokannassa suoritettavaan prosessointiin ja SQL -lauseisiin. Näiden muutokset ovat kevyempiä tehdä, sillä jollei järjestelmän tietorakennetta tarvitse muokata, ei ohjelmistopäivitystä tarvitse tehdä. Ohjelmistopäivityksen ollessa monivaiheinen jatkuvan integraation ja laadun varmistuksen ansiosta, säästyy aikaa uuden järjestelmän eduksi.

Projektissa luotu kerrosmallinen prosessointi myös helpottaa uusien tietolähteiden lisäämistä, sillä kerroksia voidaan luoda lisää, jolloin prioriteettiasteikkoa kasvatetaan ja uusi tietolähteen luoma rivi voidaan asettaa sopivalle prioriteetille datan laadun mukaisesti. Tämä helpottaa myös yrityksen sisällä tuotetun datan lisäämistä järjestelmän käyttöön datamigraatioprosessin myötä. Järjestelmään aiheutuvat datamuutokset esimerkiksi listattuihin tyyppeihin tai entiteetteihin aiheuttavat jatkossakin muutoksia sekä prosessissa käytettyihin näkymiin, siirtototeutukseen, sekä sovellukseen.

Mahdollisten järjestelmässä havaittujen datavirheiden sattuessa niiden alkuperän selvittäminen on uuden prosessin myötä helpompaa, sillä työtietokannassa data on kerätty oliokohtaisiin näkymiin, joista kerrosmallin avulla pystytään näkemään mistä tietolähteestä kukin arvo on saatu. Tämä kuitenkin vaatii työtietokannan säilyttämisen datamigraatioprosesseiden välissä, joka tuottaa lisää ylläpitokustannuksia. Aiemmassa toteutuksessa olisi täytynyt ensin tarkastaa työtietokannasta TecDoc-tietolähteen data, jonka jälkeen samaan olioon liittyvä rivi olisi haettu DriveRight-lukutietokannasta. Myös virheiden korjaaminen on helpompaa, sillä kerrosmallin avulla voidaan korjata yksi arvo lisäämällä se itse tuotettujen datojen mukana näkymään.

5.3 Käytetty aika ja saavutettu hyöty

Projektiin osallistui yhteensä kolme ohjelmistokehittäjää, jotka työskentelivät projektin parissa osin muiden työtehtävien ohessa. Projekti alkoi kerrosmalli-idean keksimisellä, jonka jälkeen sitä pivotoitiin lyhyesti noin yhden päivän ajan. Idean näyttäessä hyvältä ja toteutuskelpoiselta, alettiin valmistelemaan datan prosessointia työtietokannassa kirjoittamalla SQL-lauseita näkymien toteuttamiseksi. Varsinaista työaikaa käytettiin datan prosessoinnin luomiseen yhteensä neljä viikkoa, jonka jälkeen siirryttiin suunnittelemaan datan siirtoa järjestelmän tietokantaan. Asiaa tutkittiin noin viikko, jonka aikana tehtiin pohja vierastauluihin ja niiden avulla datan siirtämiseen sovelluksen käyttämiin tietokantatauluihin. Lopuksi tutkittiin päivityksen vaikutusta sovellukseen ja ajettiin testipäivityksiä testipalvelimelle. Projektiin käytettiin yhteensä aikaa arviolta kahdeksan työviikkoa, joka jakaantui ajallisesti kolmen kuukauden ajalle.

Ajan säästö kutakin datamigraatioprosessia kohden on uudessa järjestelmässä noin 17 tuntia. Jos ajatellaan tämä osuvan yhden vuorokauden ajalle, jolloin yksi työntekijä seuraa prosessia osa-aikaisesti, voidaan puhua yhdestä henkilötyöpäivästä. Tämä tarkoittaisi neljä kertaa vuodessa suoritettavalla päivityssyklillä neljää henkilötyöpäivää vuodessa. Aika-arvion mukaan projektiin käytettiin noin kahdeksan työviikkoa, eli 40 henkilötyöpäivää. Tällöin projekti maksaisi itsensä takaisin kymmenessä vuodessa, joten pelkästään ajan säästön kannalta projektia ei voisi sanoa kovinkaan onnistuneeksi. Kuitenkin näkymillä toteutettu migraation prosessointi vaihe on joustava, joten tulevaisuudessa on mahdollista lisätä tietolähteitä pienillä muutoksilla SQL-lauseisiin. Tämä saattaa aiheuttaa huomattavan ajan säästön, jos uusia tietolähteitä joudutaan lisäämään datamigraatioprosessiin.

Suurin hyöty saavutettiin vähentämällä sovelluspalvelimen resurssien kulutusta ja järjestelmän rasitusta datamigraatioprosessin aikana. Projektissa kehitetty tietokannan kerrosmalli on osoittautunut käytössä loistavaksi, sillä se on mahdollistanut sekä uusien tietolähteiden helpon lisäämisen että datan parantelun ja päivitysprosessin suorittamisen ketterästi, sillä data voidaan prosessoida heti tietolähteen päivityspaketin saapuessa, mutta itse järjestelmään data voidaan päivittää siihen parhaiten sopivalla hetkellä. Malli tarjoaa myös mahdollisuuden tarkastella prosessoitua dataa ennen järjestelmään siirtoa, jolloin mahdolliset datavirheet huomataan helpommin ja voidaan korjata ennen datan viemistä sovelluksen tietokantaan.

5.4 Tulevaisuuden parannuskohteita

Tulevaisuudessa tietolähteiden päivitystiheys tulee kasvamaan, joten datamigraatioprosessi tulee suorittaa kuukausittain, jos järjestelmän datasisältö halutaan pitää mahdollisimman hyvin ajan tasalla. Erityisesti tuotteiden määrä markkinoilla lisääntyy, joten jotta järjestelmä pystyisi tarjoamaan jatkuvasti uusimpia tuotteita ajoneuvomalleihin tulee tuote- ja osayhteensopivuustaulut päivittää heti, kun tietolähteiden päivitykset ovat saatavilla. Automerkit julkaisevat uusia automalleja vuosittain, jolloin ajoneuvomallinnukseen tarvittava ajoneuvodata voitanee päivittää nykyisellä tiheydellä, neljä kertaa vuodessa. Tämä voisi tarkoittaa prosessissa käytettyjen SQL-tiedostojen jakamista useampaan osaan ja useampaan transaktioon, jolloin voitaisiin päivittää helposti vain haluttu osa järjestelmän datasta. Samalla saataisiin mahdollisten virheiden vaikutukset minimoitua, sillä transaktiot olisivat pienemmissä palasissa, jolloin virhe ei aiheuttaisi koko vaiheen uusimista.

Projektin aikana dokumentointi tapahtui sekä luotuihin SQL-tiedostoihin, että tietokannan skeemoihin, näkymiin sekä tauluihin. Varsinaista vaiheittaista ohjeistusta prosessin suorittamiseen ei kirjoitettu. Dokumentoimiseen voisi käyttää jatkossa paremmin huomiota, sillä oikeiden tiedostojen löytäminen versionhallinnasta voi olla haastavaa. Seuraavan kerran datamigraatioprosessi ajettaessa voitaisiin prosessi dokumentoida yrityksen wiki-verkkosivulle. Prosessin vaiheiden dokumentoiminen myös selkeyttäisi koko datamigraatioprosessin suoritusta, sillä vaiheiden kuvausten yhteyteen voitaisiin liittää näyte sen hetkisestä datasta.

Nykyinen datamigraatioprosessi sisältää manuaalista työtä useassa vaiheessa. Automatisoinnissa voitaisiin hyödyntää Amazon AWS Lambda -palvelua, joka soveltuu ajastettuihin prosesseihin, kuten tietolähteiden lukeminen työtietokantaan [22]. Jos tietolähteet lisääntyvät uusien asiakkuuksien ja projektien myötä, tulisi prosessin automatisointia pitää korkealla prioriteetillä, jotta työntekijöiden aika voitaisiin kohdistaa tuottavampaan työhön. Ajastetut prosessit mahdollistaisivat myös resurssien lisäämisen työtietokantaan vain tarvittaessa, jolloin tietokannan ylläpitokustannukset laskisivat.

Uudistettu datamigraatioprosessi vaatii järjestelmältä noin kahden tunnin huoltokatkon datasiirron ajaksi. Tämä voitaisiin tulevaisuudessa välttää, jos järjestelmän saisi huoltotilaan, jolloin sovellus tarjoaisi vain rajoitetusti palveluita. Tällöin voitaisiin kytkeä ajoneuvomallinnukseen ja varaosien tarjoamiseen liittyvät palvelut pois päältä hetkellisesti, jolloin sovellus toimisi muutoin normaalisti. Vaihtoehtoisesti sovellus voitaisiin jakaa useampaan moduuliin käyttäen modulaarista arkkitehtuuria *valverde2003hierarchical*. Muutos antaisi mahdollisuuden asettaa päivitettävä

moduuli huoltotilaan muun järjestelmän toimiessa normaalisti. Tämä on kuitenkin prioriteetiltaan pieni parannus, sillä datasiirto voidaan ajoittaa aikaan, jolloin järjestelmän käyttöaste on matalimmillaan. Tätä tukee ajatus datamigraatioprosessin automatisoinnista.

6. YHTEENVETO

Työssä käsiteltiin projektia, jonka aikana ADA Drive Oy:n BASE-tuotteen mallinnusdatan datamigraatioprosessia pyrittiin uudistamaan ja tehostamaan. Mallinnusdata koostetaan useasta tietolähteestä, jotka tarjoavat päivityksiä vähintään neljä kertaa vuodessa, jonka vuoksi datamigraatioprosessi on toistuva ja kriittinen osa järjestelmän ylläpitoa, joka takaa järjestelmän toiminnallisuuden ja mallinnuksen tarkkuuden. Ennen projektin alkua käytössä oli useaan eräajoon perustuva datamigraatio, jonka heikkoutena oli toistettavuus, pitkä ajoaika ja järjestelmän suuri ajonaikainen kuormittavuus.

Datamigraatioprosessin parantamiseksi kehitettiin uudenlainen migraatioprosessi, joka perustuu PostgreSQL-tietokantaan. Työtietokannassa suoritetaan datan prosessointi ja laadun tarkkailu. Prosessointi tapahtuu tietokannan omilla funktioilla ja ominaisuuksilla, kuten näkymillä ja merkkijonofunktioilla, joilla voidaan suorittaa taulujen datamallin muutokset ja datan tyypitys. Työtietokannassa useista tietolähteistä peräisin oleva data yhdistetään prosessoinnin kautta valmiisiin näkymiin, joista se voidaan siirtää sovellusjärjestelmän tietokannan tauluihin. Siirto työtietokannasta järjestelmätietokantaan toteutettiin hyödyntämällä PostgreSQL:n vierastyötauluja, jolloin järjestelmän sovelluspalvelin ei kuormitu siirtoprosessissa.

Datan prosessointia varten kehitettiin tietokantaan kerrosmalli, jonka avulla eri datalähteistä peräisin olevat datat pystyttiin priorisoimaan ennen niiden yhdistämistä. Kerrosmallissa kunkin tietolähteen yhteen objektiin liittyvä data koottiin useaksi allekkaiseksi kerrokseksi. Kerroksille asetettiin tietolähteen laadun mukainen prioriteettiarvo, jonka mukaan kerrokset litistettiin yhdeksi riviksi, joka sisälsi parhaan mahdollisen yhdistelmädatan kaikista tietolähteistä. Kehitetty malli mahdollisti datan laadun tarkkailun ja jäljitettävyyden järjestelmän tietokannasta takaisin tietolähteeseen.

Datamigraatioprosessin uudistus lyhensi huomattavasti prosessiin käytettävää aikaa ja vähensi resurssien kulutusta. Se paransi prosessin toistettavuutta ja ylläpidettävyyttä, sillä SQL-pohjainen datan prosessointi on helposti muokattavissa tulevaisuutta silmällä pitäen. Uusien tietolähteiden käyttöönotto on jatkossa helpompaa

ja ketterämpää, kunhan tietolähteet ovat käytettyyn relaatiotietokantaan sopivassa muodossa.

LÄHTEET

[1] R. J. Rittinghouse John, *CLOUD COMPUTING: Implementation, Management, and Security.* CRC Press, Taylor & Francis Group, Boca Raton, FL, 2010.

[2] A. D. Oy, "Ada drive oy -kotisivut," 2016. Available (15.12.2016): `http://www.adadrive.fi`.

[3] Trafi, "Liikenteen turvallisuusvirasto," 2016. Available (15.12.2016): `http://www.trafi.fi/`.

[4] Bisnode, "Bisnode infotorg," 2016. Available (16.12.2016): `http://www.infotorg.se/`.

[5] S. vegvesen, "Statens vegvesen," 2016. Available (16.12.2016): `http://www.vegvesen.no/`.

[6] TecAlliance, "Tecdoc catalogue data package," 2016. Available (16.12.2016): `https://www.tecalliance.net/en/products/tecdoc-catalogue-data-package/`.

[7] T. M. Somers and K. Nelson, "The impact of critical success factors across the stages of enterprise resource planning implementations," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, pp. 1–10, IEEE Computer Society, 2001.

[8] N. F. Albrecht Alexander, "Managing etl processes," 2008.

[9] A. Behm, A. Geppert, and K. R. Dittrich, *On the migration of relational schemas and data to object-oriented database systems.* Citeseer, 1997.

[10] T. P. G. D. Group, "Postgresql documentation," 2016. Available (25.10.2016): `https://www.postgresql.org/docs/9.5/static/index.html`.

[11] Microsoft, "Sql server 2016 technical documentation," 2017. Available (2.4.2017): `https://opbuildstorageprod.blob.core.windows.net/output-pdf-files/en-us/SQL.sql-content/live/sql-server.pdf`.

[12] Oracle, "Oracle database 12c release 2," 2017. Available (2.4.2017): `http://docs.oracle.com/database/122/index.htm`.

[13] M. T. Minella, *Pro Spring Batch.* Apress, 1 ed., 2011.

[14] R. T. K. R. G. D. L. W. M. M. S. C. H. G. Ward Lucas, Syer Dave, "Spring batch - reference documentation," 2014. Available (25.10.2016): `http://docs.spring.io/spring-batch/reference/html/`.

[15] E. Amazon, "Amazon web services," *Available in: http://aws. amazon. com/es/ec2/(November 2012)*, 2015.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[17] G. Reese, *Cloud Application Architectures*. O'Reilly Media, Inc., 1 ed., 2009.

[18] A. W. Services, "Amazon elastic compute cloud," 2016. Available (13.10.2016): `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf`.

[19] DriveRight, "Driveright data," 2015. Available (4.4.2017): `http://driverightdata.com/about/`.

[20] DriveRight, "Tyres fitment database," 2015. Available (4.4.2017): `http://driverightdata.com/products/fitment-data/`.

[21] DriveRight, "Wheel and tyre visualizer," 2015. Available (4.4.2017): `http://driverightdata.com/products/wheel-tyre-visualizer/`.

[22] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, *et al.*, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, pp. 179–182, IEEE, 2016.